슈퍼컴퓨터를 위한 고성능 스토리지 활용

KISTI 슈퍼컴퓨팅센터

목 차

표 목차

그림 목차

제 1 장 서 론

최신형 슈퍼컴퓨터인 IBM p690 시스템을 도입 설치하는 시점에서 본 시스템의 성능에 걸맞고 기존의 슈퍼컴퓨터들을 활용하여 얻어낸 소중한 데이터를 안전하고 효율적으로 관리하기 위한 대용량 스토리지 시스템(High Performance Stroage System :HPSS)을 도입하여 운영 중에 있습니다. HPSS에는 전 세계 유수의 슈퍼컴퓨터 센터에서 그들의 데이터를 안전하게 관리하기 위하여 널리 쓰이고 있는 시스템입니다. 기존의 어떤 데이터 저장 시스템과도 비교할 수 없는 고성능 시스템이며 전 세계적 추세인 그리드 구축 부문에서 데이터 그리드 구축용 시스템으로 그 가능성을 높게 평가받고 있습니다. KISTI에 설치된 HPSS 시스템은 최신 4.3 버전이며 1대의 core 서버와 데이터를 실제적으로 처리하는 mover 8대로 구성되어 있습니다. 약 3.6TB의 디스크 캐시와 380TB의 LTO 테이프로 구성되어 있는 HPSS 시스템은 컴퓨팅 작업을 통하여 얻은 어떠한 크기의 결과물도 효율적으로 처리하며 안전하게 보관하는 성능을 보장합니다. 사용법은 단순하고 직관적이며 일반 유닉스 사용 경험이 있는 사용자라면 누구나 쉽게 이용할 수 있습니다.

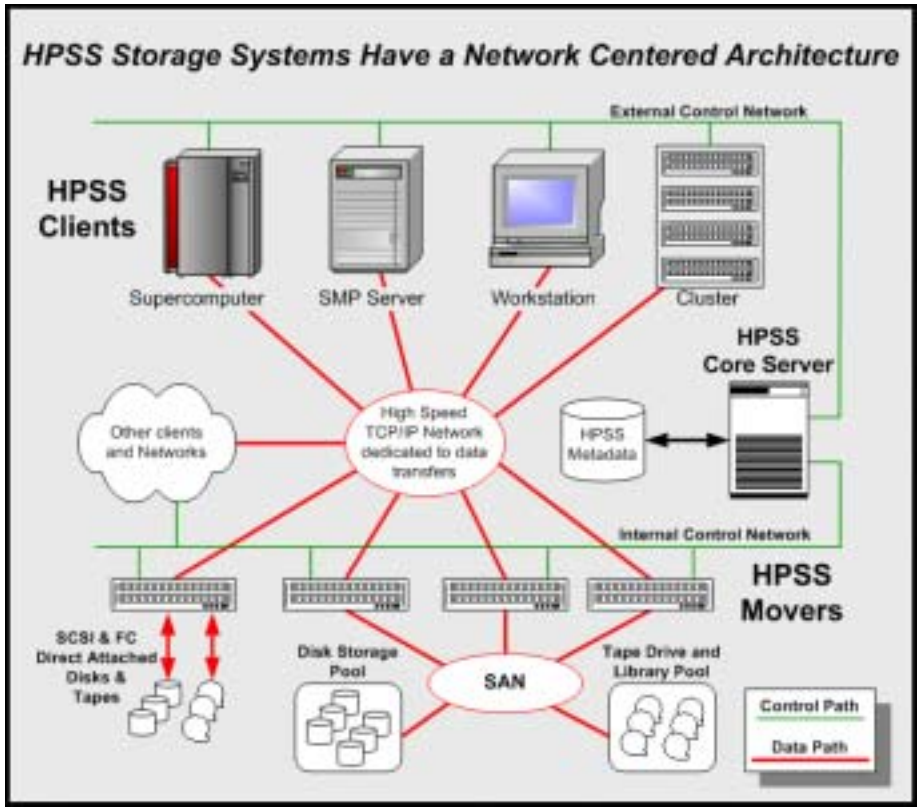
본 지침서는 HPSS 사용을 원활하게 하게 위하여 일반 사용자를 대상으로 작성되었으며 보다 깊이 있는 사용법을 알고 싶은 사용자는 추후에 KISTI 웹사이트를 통하여 보다 상세한 정보를 제공받을 수 있습니다.

<그림 1-1> 전 세계 HPSS 사이트

제 2 장 HPSS(High Performance Storage System)

2.1 HPSS의 개요

HPSS의 키포인트는 일반적인 스토리지 시스템의 단일 서버 프로세서가 아닌 네트워크이다. HPSS는 소켓 인터페이스를 가지는 모든 네트워크 환경을 지원한다. TCP/IP 뿐만 아니라 Fibre Channel Standard(FCS)와 ATM도 지원한다. 사용자는 사용자 인터페이스 유틸리티에 의해 선택된 서버노드로 데이터를 요청하면 HPSS 서버는 직접적으로 데이터를 읽고 쓰기 위한 network-attached 스토리지 디바이스를 지정한다. 결과적으로 high-speed 데이터 전송 네트워크를 통하여 병렬적으로 모든 작업을 수행하게 된다. 데이터를 읽을 때의 실제 스피드는 어디에 데이터가 저장되어 있는지(faster for disk, slower for tape) 그리고 어떤 종류의 네트워크를 사용자가 사용하는지에 달려 있다.



<그림 1-2> HPSS의 Network centered 구조

쓰기 속도는 앞서 말한 두 가지 factor 외에 데이터의 Class of Service(COS)에 연관되어 있다. COS는 특별히 그룹화된 리소스들과 그들의 configuration을 지칭하는 HPSS에서 사용되는 용어이다. HPSS는 COS 메카니즘을 이용하여 파일들을 정책적 set을 기반으로 분류하고 있다. 일단 파일들이 HPSS로 전송되어 오면 이것들은 archival system에 저장된다. 개인용 워크스테이션이나 슈퍼컴퓨터들과는 다르게 archival system은 파일들을 여러 디바이스들로 구성된 dynamic hierarchies내에 저장한다. 파일들은 처음에는 그 계층 구조 중 highest level(disk)에 저장된다. 파일들은 특정 기간동안 디스크에 저장된 후 또는 사용자의 명령어에 의해 계층 구조 중 lowest level(tape)로 migration 된다. 파일들이 tape로 복사된 후에 이것들을 다른 파일들의 생성과 수정을 하기 위한 공간 확보를 위하여 디스크로부터 삭제될 수 있다. 사용자가 파일에 접근 했을 때 그것이 만일 디스크에 존재하지 않는다면 그 파일은 tape에서부터 disk로 migration 된다. 그리고 나서 요청된 호스트로 전송되게 된다. HPSS와 같은 archival storage system은 매우 안정적으로 huge volume들을 관리하며 수백 만개의 파일들이 HPSS에 일정기간동안 저장될 수 있다. NFS나 다른 물리적으로 연결된 디스크와 같은 일반적인 시스템으로는 HPSS가 대규모의 파일들을 관리하는 것 같은 기능을 감당하기는 어렵다.

- HPSS는 대규모 스토리지 환경을 위한 계층적 스토리지 관리(HSM) 및 서비스 제공을 위한 소프트웨어이며 수백 테라 바이트에서 수 petabyte까지의 데이터를 처리할 수 있다.

- HPSS는 고성능 컴퓨터나 디스크, 테이프 라이브러리들 사이에 매우 큰 데이터 objects를 이동시키도록 고안되었으며 처리속도는 단지 컴퓨터나 네트워크, 스토리지 장치 등의 최대 속도에 의해서만 제한된다.

- HPSS는 초당 수백 megabyte의 속도로 멀티 network-connected 디스크들로부터 병렬 데이터 전송 처리할 수 있다.

- HPSS는 전 세계의 수많은 연구소에서 사용되고 있으며 응용범위는 고 에너지 물리학, 슈퍼컴퓨터 센터, 과학 데이터 처리 등으로 다양하다. HPSS는 미국 국립 슈퍼컴퓨터 센터들과 산업체들 간의 협력에서 나온 결과물이며 IBM에 의해 상업적으로 배포되고 있다.

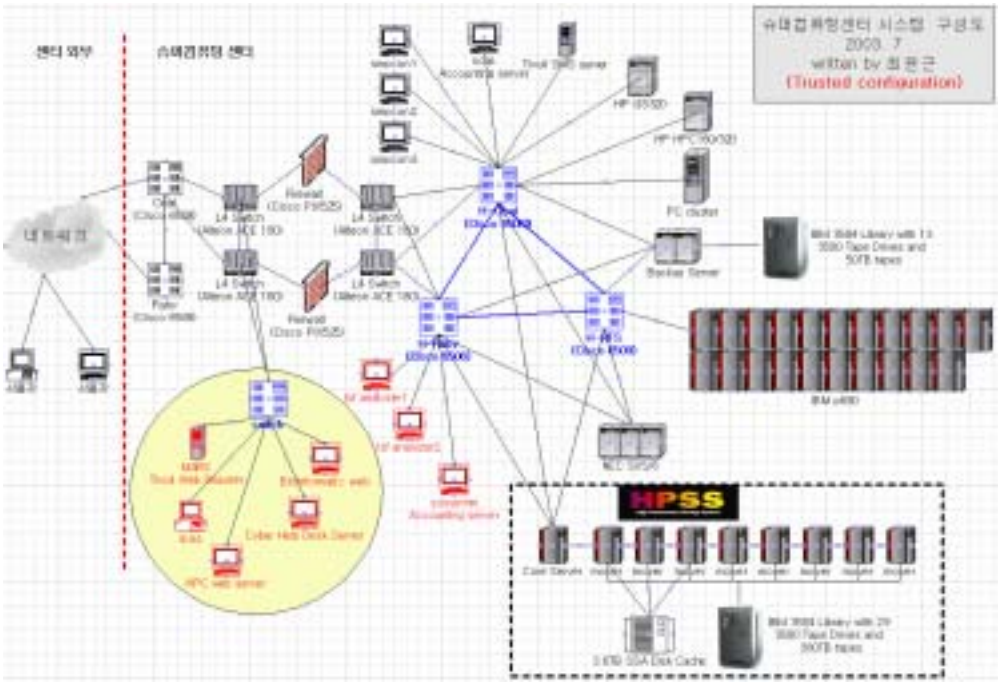
- HPSS의 중요한 목적은 network-centered한 시스템과 분산 계층적 관리 시스템으로부터 발생하는 대량의 대규모 파일들을 원활히 처리하는데 있다.

- HSM으로 대표적인 Tivoli와의 성능 비교
  (참조 : http://www4.clearlake.ibm.com/hpss)

<표 1-1> HPSS와 Tivoli의 성능비교

구 분	HPSS	Tivoli
스토리지 능력	1PB 이상	수백 TB
처리량	. >50MB/sec for a single processing unit . >1GB/sec using parallel processing	. >18MB/sec for a single processing unit

2.2 HPSS와 슈퍼컴퓨팅센터 시스템 구성도



<그림 1-3> System configuration with HPSS

2.3 HPSS 리소스 관리정책

2.3.1시스템 운영 계획

□ 파일 시스템

○ 파일 시스템은 슈퍼컴퓨터 기종에 따라 다음과 같이 나누어 짐

- HP(구 Compaq) 사용자 : /COMPAQ/user_id
- NEC 사용자 : /NEC/user_id
- PC cluster(구 Cray) 사용자 : /CRAY/user_id
- IBM 사용자 : /IBM/user_id

○ 사용자는 HPSS 시스템에서 위의 4개 파일시스템 중 하나로 홈 디렉터리를 가지게 됨

2.3.2 사용자 인터페이스

□ 사용법은 <3. 사용자 인터페이스> 참조

○ 병렬 ftp

- pftp_client(사용자 홈 디렉터리 또는 스크래치 디렉터리 등 어떤 location 에서 실행 가능)를 이용하여 HPSS 시스템에 접속
- 호스트 이름은 core, 접속 포트는 4021
- 접속 예 : %pftp_client core 4021

○ HSI

- HSI를 이용하여 HPSS 시스템에 접속
- 접속 예 : %hsi

○ Client API

- NFS는 서비스하지 않음

2.3.3 계정 정책

□ KISTI 슈퍼컴퓨터 사용자 계정과 동일한 ID의 계정

□ 패스워드는 슈퍼컴퓨터 계정의 것과 동일 또는 다를 수 있음

□ HPSS 계정의 패스워드는 사용자가 임의로 바꿀 수 없음(슈퍼유저 only)

□ 계정신청 관련 : <3. 사용자 인터페이스> 참조

제 3 장 사용자 인터페이스

3.1 HPSS 계정 신청

HPSS 서비스를 이용하려면 HPSS 시스템에 access 할 수 있는 계정이 별도로 필요하다. 슈퍼컴퓨터 시스템의 계정을 가지고 있는 사용자는 HPSS계정을 다음의 e-메일로 신청한다.

> e-mail : jangoq@kisti.re.kr
> phone : 042-869-0540

현재 HPSS 사용자 계정의 패스워드를 변경할 수 있는 방법은 보안상의 이유로 HPSS 시스템 관리자(위)에게 변경을 요청하는 방법만으로 서비스가 제공되고 있다. 즉 주어진 패스워드를 사용자가 임으로 변경할 수 없다.

3.2 HPSS 사용을 위한 기본 지침

HPSS는 사용자들에게 다양한 유저 인터페이스를 제공한다. 그 중 일반 계정 이용자가 가장 쉽게 이용할 수 있고, HPSS의 장점을 가장 잘 살릴 수 있는 방법이 pftp_client이다. pftp_client는 일반 ftp와 유사한 명령어 체계를 가지고 있지만, 일반 ftp와는 달리 KISTI의 슈퍼컴퓨터들과 HPSS 서버들간의 병렬 데이터 전송을 가능하게 함으로써 데이터 전송의 속도를 높일 수 있다.

또한 pftp_client는 슈퍼컴퓨터와 HPSS 디스크 케쉬 간의 직접 데이터 전송을 가능하게 함으로써 HPSS 서버에 부하를 최소로 유지하도록 한다. HPSS는 pftp_client 외에도 HSI, client API(Application Program Interface) 등을 제공한다. HPSS는 API는 일반 사용자가 프로그램을 통하여 HPSS 시스템과 데이터를 주고받을 수 있는 기능을 제공한다.

> 사용자 인터페이스를 사용하기 전 HPSS 시스템으로 전송할 데이터들은 “tar" 유틸리티를 이용하여 한 파일로 묶은 후 전송하기를 강력히 권고한다. HPSS는 대규모의 단일 파일을 처리하는 것은 유리하지만 소규모의 대량파일들을 처리 하는 데는 효율성 면에서 떨어진다. ”tar" 사용법은 기타 유닉스 매뉴얼을 참조한다.

3.3 병렬 ftp(pftp_client) 사용법

HPSS의 대표적인 사용자 인터페이스인 pftp_client는 ftp와 그 사용법과 기능 등이 유사하나 병렬분산처리 기능을 가지고 있는 점이 일반 ftp의 기능과 다른 점이다.

<부록 3>에서 볼 수 있는 HPSS 성능 테스트 결과 및 ftp와의 비교성능 실험에서도 pftp_client는 월등한 성능을 나타낸다. 물론 일반적인 ftp로도 HPSS로 접속할 수 있다. HPSS에서 제공하는 ftp 인터페이스는 일반 ftp와 동일한 명령어와 사용법을 이용하지만, HPSS pftp와 비교하였을 때 효율성 및 이용의 편리성 측면에서 권장하지 않는다. 그러므로 여러 가지 성능과 기능 면에서 볼 때 HPSS의 사용자 인터페이스로 pftp_client의 사용을 강력히 권장한다.

3.3.1 pftp_client를 이용한 접속

□ 로그인

○ HP SMP(frontsmp), Tera cluster, IBM p690 시스템의 계정을 가지고 있는 사용자는 각 시스템에 로그인 한 후 다음 명령어를 수행한다.

```
$ pftp_client core 4021
```

○ NEC 사용자는 홈에 있는 .cshrc 파일을 다음과 같이 수정한 후 logout 한 다음 다시 login 해서 위의 명령어를 사용한다.

```
%vi .cshrc
  첫줄에 중에서 다음을 수정한다.
set path = ( /sbin /usr/sbin /etc /usr/bin /usr/ucd .) 을
set path = ( $path /sbin /usr/sbin /etc /usr/bin /usr/ucd .)
로 수정한 후 저장한다.
```

○ 로그인 후 다음과 같은 메시지를 볼 수 있다.

```
Parallel block size set to 8388608.
PdataSockBufSize reset equal or below sb_max (1048576)
Connected to core.hpcnet.ne.kr.
220-
220-************************************************
220-*                                              *
220-* This is the HPSS 4.3 system running at KISTI  *
220-*                                              *
220-************************************************
220-
220 core FTP server (HPSS 4.3 PFTPD V1.1.1 Mon Dec 3 14:30:48
KORST 2001) ready.
Name (core:ykchoi):
```

○ "Name (core:ykchoi) : " 에 HPSS administrator로부터 받은 ID를 입력한다. 엔터를 친 후 다시 암호 입력 메시지가 나오면 암호를 입력한다.

```
Name (core:ykchoi): ykchoi98
331 Password required for /.../kisti_cell.hpcnet.ne.kr/ykchoi98.
Password:
```

○ 정확히 HPSS 계정 ID와 패스워드를 입력하면 다음의 메시지를 볼 수 있다.

```
230 User /.../kisti_cell.hpcnet.ne.kr/ykchoi98 logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
**** NOTE: Server supports Parallel Features     ****
****         Auto-Parallel Substitution Enabled.  ****
**** NOTE: Protocol set to PDATA_AND_MOVER        ****
Multinode is Disabled.
ftp>
```

○ 230으로 시작되는 첫 번째 줄을 보게 되면 자신의 HPSS 계정 ID를 볼 수 있다. (예제의 경우 "ykchoi98").

○ pwd 명령어를 이용하면 현재 HPSS 내의 자신의 디렉터리를 볼 수 있다.

```
ftp> pwd
257 "/home/ykchoi98" is current directory.
ftp>
```

작업 디렉터리는 사용자 계정에 따라 다를 수 있다. 사용자는 작업 영역에 따라“home/ykchoi98"의 서브디렉토리를 만들 수 있다(mkdir 명령어 사용). 예를 들어 ”test"라는 서브디렉토리를 만들고자 할 때는 HPSS 내에서 mkdir test 명령어를 사용한다. 작업 공간의 이동은 “cd dir_name” 명령어를 이용한다. ftp> 프롬프트 상태에서 로컬(자신의 컴퓨터를 의미한다)의 디렉터리를 이동하려면 “lcd dir_name” 명령어를 이용한다.

```
ftp> mkdir test
257 MKD command successful.
ftp> cd test
250 CWD command successful.
ftp> pwd
257 "/home/ykchoi98/test" is current directory.
ftp>
```

□ Parallel file store - pput

○ pput 는 로컬로부터 HPSS로 파일을 전송하는 명령어이다.

```
ftp> pput file_name                    // HPSS 홈 디렉터리에 저장
또는
ftp> pput file_name1 file_name2
// HPSS 홈 디렉터리에 다른 이름으로 저장
또는
ftp> pput file_name1 /home/myhome/file_name1
// HPSS 홈 디렉터리가 아닌 사용자 지정 디렉터리에 저장
```

○ 파일을 전송하기 전에 자신의 로컬 디렉터리에 있는 파일을 확인해 본다. !ls 명령어를 이용한다.

```
ftp> !ls
300.out                    pput.ksh
babo.txt                   pput.out
frontsmp_pput_1.txt        pput_final.txt
mksparse                   pput_frontsmp.new.txt
pget.ksh                   pput_frontsmp.txt
ftp>
```

○ HPSS로 전송할 파일을 정했으면 pput 명령어를 사용한다. 여기서는 300.out 파일을 전송해 보기로 한다.

```
ftp> pput 300.out
local: 300.out remote: 300.out
200 Command Complete.
200 PORT command successful.
150 Opening BINARY mode data connection for 300.out.
226 Transfer complete.
342635 bytes sent in 0.08 seconds (4.29 Mbytes/s)
ftp>
```

○ ls 명령어를 사용하면 HPSS로 파일이 전송된 것을 확인할 수 있다.

```
ftp> ls
200 PORT command successful.
150 Opening ASCII mode data connection for file list.
300.out
medium
medium_Merkava
test
226 Transfer complete.
39 bytes received in 0.01 seconds (4.88 Kbytes/s)
ftp>
```

○ 여러 개의 파일을 전송할 때는 mpput 명령어를 사용한다. 주의할 점은 interactive mode를 off 하기 위하여 prompt 명령어를 mpput을 사용하기 전에 사용한다. 이 모드가 on 상태를 유지하게 되면 파일이 한번씩 전송될 때마다 확인 메시지가 나와서 사용자는 계속 y 키를 눌러야 한다.

```
ftp> prompt
Interactive mode off.
ftp> mpput file_names1 file_name2 file_nameN
```

파일의 개수가 많지 않을 경우 파일의 이름을 위와 같이 나열할 수 있으마 와일드 카드를 이용하면 보다 편리하다

```
ftp> mpput *.dat
```

□ Parallel file retrieval - pget

○ pget 는 HPSS로부터 로컬로 파일을 전송하는 명령어이다.

```
ftp> pget file_name
```

○ 파일을 전송하기 전에 자신의 HPSS 디렉터리에 있는 파일을 확인해 본다. ls 명령어를 이용한다.

```
ftp> ls
200 PORT command successful.
150 Opening ASCII mode data connection for file list.
300.out
226 Transfer complete.
9 bytes received in 0.01 seconds (1.29 Kbytes/s)
ftp>
```

○ 로컬로 전송할 파일을 정했으면 pget 명령어를 사용한다. 여기서는 300.out 파일을 전송해 보기로 한다.

```
ftp> pget 300.out
local: 300.out remote: 300.out
200 PORT command successful.
150 Opening BINARY mode data connection for 300.out (342635
    bytes).
226 Transfer complete.
342635 bytes received in 1.62 seconds (205.91 Kbytes/s)
ftp>
```

○ !ls 명령어를 사용하면 자신의 로컬 디렉터리에 파일이 전송된 것을 확인할 수 있다.

○ 여러 개의 파일을 전송할 때는 mpget 명령어를 사용한다.

```
ftp> prompt
Interactive mode off.
ftp> mpget file_names
```

□ 기타 명령어 - bye, del, mdel

○ pftp_client 종료할 때는 bye 명령어를 이용한다.

```
ftp> bye
421 Timeout (900 seconds): closing control connection.
frontsmp [/system/super/ykchoi] $
```

○ HPSS에 전송된 파일을 삭제할 경우 del 명령어를 사용한다.

```
ftp> del pput.txt
250 DELE command successful.
ftp>
```

○ 여러 개의 파일을 한꺼번에 삭제할 경우 mdel을 사용한다. interactive mode를 off 하기 위하여 prompt 명령어를 mdel을 사용하기 전에 사용한다. 이 모드가 on 상태를 유지하게 되면 파일이 한번씩 전송될 때마다 확인 메시지가 나와서 사용자는 계속 y 키를 눌러야 한다.

```
ftp> mdel *.txt
mdel babo.txt? y
250 DELE command successful.
mdel frontsmp_pput_1.txt? y
250 DELE command successful.
mdel pget_final.txt? y
250 DELE command successful.
mdel put_1030.txt?
```

○ prompt 사용 후

```
ftp> prompt
Interactive mode off.
ftp> mdel *.txt
250 DELE command successful.
250 DELE command successful.
250 DELE command successful.
250 DELE command successful.
250 DELE command successful.
ftp>
```

○ ftp> 상태에서 사용할 수 있는 모든 명령어를 보려면 ?를 사용한다

```
ftp> ?
```

□ ls -lh

○ -lh 옵션은 저장된 파일의 COS #를 볼 수 있다. 다음 예의 경우는 1100이다

```
ftp> ls -lh
200 PORT command successful.
150 Opening ASCII mode data connection for file list.
-rw-r-----    1 110      2001    120 Mar  7 09:35 hpsstest.out
-rw-r-----    1 110      2001     30 Mar  7
 09:40hpsstest.out.020304
226 Transfer complete.
ftp>
```

□ pftp_client 명령어

기능	명령어	설명
ACCESS HPSS	**pftp_client**	begin interactive pftp session
TERMINATE pftp	**quit**	end interactive pftp session
	bye	end interactive pftp session
SAVE/UPDATE A FILE	**pput** *filename*	save/update filename
	pput *flename* suba/*filename*	save/update filename in subdirectory suba
	mpput *.*ext*	save/update all files with extension ext
APPEND TO A FILE	**append** *filename hpss-file*	append filename to hpss-file
RETRIEVE A FILE	**pget** *filename*	retrieve filename
	cd suba, **pget** *filename*	retrieve filename from subdirectory suba
	mpget *.*ext*	retrieve all files with extension ext
DELETE A FILE	**delete** *filename*	delete filename
	del *filename*	delete filename
	delete suba/*filename*	delete filename from subdirectory suba
	mdelete suba/*.*ext*	delete all files with extension ext from subdirectory suba
TOGGLE PROMPT	**prompt**	toggle interactive prompt for mget/mput/mdelete
LIST FILES	**ls**	list files
	ls -l or **dir**	list files and file information
	ls *suba*	list files in subdirectory suba
CHANGE FILENAME	**rename** *oldname newname*	change oldname to newname
SUBDIRECTORY COMMANDS	**mkdir** *subdir-name*	add subdir-name
	mkdir suba/*subdir-name*	add subdir-name beneath subdirectory suba
	rmdir *subdir-name*	delete subdir-name
	pwd	print working directory

□ HPSS 사용자 매뉴얼(영문)

○ 보다 자세한 pftp_client 사용법은 다음 사이트에서 다운 받을 수 있다.

http://www4.clearlake.ibm.com/hpss/index.jsp

3.3.2 배치작업에서의 pftp_client 사용

□ 개요

배치작업에서의 HPSS 사용은 배치 스크립트 내에 간단한 코드를 삽입하는 것으로 간편하게 이용할 수 있다.

□ 배치 스크립트에 삽입할 코드

배치 스크립트에 대한 자세한 내용은 각 시스템의 사용자 가이드를 참조한다.

```
#이 부분은 배치 스크립트의 종류에 따라 달라진다.
#다음 예는 IBM의 배치 스크립트(LoadLeveler)의 경우이다
#!/bin/ksh
#@ job_type = serial
#@ notification = always
#@ notify_user =  wjnadia@kisti.re.kr
#@ class = normal
#@ resources = ConsumableCpus(1)
#@ queue
#위의 배치 스크립트 종류에 관계없이 다음 코드를 삽입한다.
pftp_client -vn << -EOF
user hpss_id hpss_pw    #HPSS 계정(아이디, 패스워드) 입력
mkdir dir1              #저장하고자 하는 디렉터리를 만든다(옵션)
cd dir1                 # 만든 디렉터리로 이동(옵션)
pput files              # 원하는 파일을 전송
cd ...
mkdir dir2
cd dir2
pput file2
quit
EOF
#주의할 것은 이 스크립트를 돌리는 디렉터리에 저장하고자 하는 파일이
  있어야 한다. 예를 들어 /a/b에서 스크립트를 돌리는데 /a/b/c에 있는
  파일을 전송하려고 pput /a/b/c/file하면 작동하지 않는다.
```

3.4 HSI 사용법

pftp_client와 더불어 많이 사용되는 HSI는 pftp_client와 성능은 비슷하나 좀더 편리한 옵션들을 제공한다. 예를 들어 한 디렉터리를 지정하여 그 하위 디렉터리 및 파일을 모두 전송할 수 있는 옵션을 가지고 있다.

3.4.1 HSI를 이용한 접속

□ 로그인

○ HP SMP(frontsmp, whiparam), pc cluster, IBM p690 및 NEC 시스템의 계정을 가지고 있는 사용자는 각 시스템에 로그인 한 후 다음 명령어를 수행한다.

```
$ hsi
```

○ 로그인 후 다음과 같은 메시지를 볼 수 있다(다음은 nobel에서의 예제 이다).

```
nobela:/system/super/hpssadm> hsi
[connecting to core.hpcnet.ne.kr/1217]
DCE Principal: hpss
Password:
```

○ 패스워드를 입력하면

```
nobela:/system/super/hpssadm> hsi
[connecting to core.hpcnet.ne.kr/1217]
DCE Principal: hpss
Username: hpss  UID: 210  CC: 210 Copies: 1 [hsi.2.8 Sat Jun 28
03:17:02 KORST 2003]
?
```

○ “?”는 hsi 명령 프롬프트이며 위와 같이 나온 후에 hsi 명령어를 사용할 수 있게 된다.

□ 로그인 환경(프롬프트)

○ hsi 명령 프롬프트인 “?”를 간단한 config 파일을 만들어서 좀더 보기 편리하게 바꾸어 보도록 하자.

○ 다음을 에디터로 타이핑하여 .hsirc 라는 이름으로 저장한 후 사용자 홈에 놓는다. 그리고 로그아웃 후 다시 로그인 한 다음 hsi로 로그인해 보자.

```
global:
copies = 1
PS1 = "%w2->"
promptlen = 50
promptdirlen = 30
PS2 = "[more] "
autobackup = off
drive = Z:
```

○ hsi 명령 프롬프트가 보기 좋게 바꿔져 있다.

```
nobela:/system/super/hpssadm> hsi
[connecting to core.hpcnet.ne.kr/1217]
DCE Principal: hpss
Username: hpss  UID: 210  CC: 210 Copies: 1 [hsi.2.8 Sat Jun 28
03:17:02 KORST 2003]
/home/hpss->
```

□ 로그인 환경(자동 로그인) - keytab 파일

아이디나 패스워드 없이 자동 로그인 되게 하기 위해서는 .hsirc 파일에 다음을 추가한다. 자동 로그인은 배지작업을 할 때 특히 유용하게 사용된다 (pftp_client 사용 시 .netrc파일과 유사한 역할).

```
global:
copies = 1
PS1 = "%w2->"
promptlen = 50
promptdirlen = 30
PS2 = "[more] "
autobackup = off
site = kisti.prod        # 추가
drive = Z:
```

```
kisti.prod:
drive = C:
host = core.hpcnet.ne.kr
port = 1217
authmethod = keytab
keytab = %H/.private/.ktb_file
# .ktb_file은 HPSS 관리자만이 만들 수 있다(사용자가 만들 수 있는 것
  이 아님. HPSS 관리자에게 요청해야 함)
principal = myuser,keytab
# 위의 "myuser" 는 HPSS 아이디 이며 사용자는 자신의 아이디로 바꾼다.
```

그리고 .ktb_file을 HPSS 관리자에게 요청하여 발급받아야 한다. 이 파일에는 사용자 정보가 암호화되어 저장되어 있다. .ktb_file은 물론 다른 이름을 가질 수 있다. 그러나 파일 내임의 일관성을 위해서 .ktb_username으로 발급되어진다. 그리고 중요한 것은 이 파일은 사용자 홈이 아닌 .private 디렉터리에 놓여져야 한다. 이를 위해서 .private 디렉터리를 다음과 같이 만든다.

```
$pwd
/home/user
$mkdir .private
$cd .private
$pwd
/home/user/.private
```

/home/user/.private에 .ktb_username 파일을 복사해 놓고(이전에 HPSS 관리자에게 요청하여 이 파일을 확보한다) 파일 모드를 600으로 바꾼다.

```
%chmod 600 .ktb_username
```

3.4.2 HSI 사용법

□ Parallel file store - put

<주의> pftp_client의 pput과 혼동하지 말것

○ put 는 로컬로부터 HPSS로 파일을 전송하는 명령어이다.

```
/home/hpss-> put file_name    // HPSS 홈 디렉터리에 저장
  또는
/home/hpss-> put file_name1 file_name2
// HPSS 홈 디렉터리에 다른 이름으로 저장
  또는
/home/hpss-> pput file_name1 /home/myhome/file_name1
// HPSS 홈 디렉터리가 아닌 사용자 지정 디렉터리에 저장
```

○ 파일을 전송하기 전에 자신의 로컬 디렉터리에 있는 파일을 확인해 본다. !ls 명령어를 이용한다.

```
/home/hpss-> !ls
```

○ HPSS로 전송할 파일을 정했으면 put 명령어를 사용한다. 여기서는 compaq.tar 파일을 전송해 보기로 한다.

```
/home/hpss-> put compaq.tar
put compaq.tar : /home/hpss/compaq,tar ( 8611840 bytes, 5741.3
KBS (cos=111))
/home/hpss->
```

○ ls 명령어를 사용하면 HPSS로 파일이 전송된 것을 확인할 수 있다.

```
/home/hpss-> ls
/home/hpss:
compaq.tar
/home/hpss->
```

○ 여러 개의 파일을 전송할 때는 pftp_client처럼 별도의 명령어(mpput)가 있는 것이 아니고 put과 와일드 카드(?, * 등)를 이용 한다.

```
/home/hpss-> put *.c
```

파일의 개수가 많지 않을 경우 파일의 이름을 아래와 같이 나열할 수 있다.

```
/home/hpss-> put a.c b.c d.dat
```

○ HSI는 pftp_client가 제공하지 않는 편리한 기능이 있는데 디렉터리 명을 지정하면 그 하위 디렉터리 및 파일들을 통째로 전송할 수 있다. 다음과 같이 -R 옵션을 이용한다.

```
/home/hpss-> put -R dir_name
```

□ Parallel file retrieval - get

○ get 는 HPSS로부터 로컬로 파일을 전송하는 명령어이다.

```
/home/hpss->get file_name
```

○ 파일을 전송하기 전에 자신의 HPSS 디렉터리에 있는 파일을 확인해 본다. ls 명령어를 이용한다.

```
/home/hpss-> ls
/home/hpss:
compaq.tar
/home/hpss->
```

○ 로컬로 전송할 파일을 정했으면 get 명령어를 사용한다. 여기서는 compaq.tar 파일을 전송해 보기로 한다.

```
/home/hpss-> get compaq.tar
```

○ -R 옵션을 이용하여 디렉터리 명을 지정하면 그 하위 디렉터리 및 파일들을 통째로 전송할 수 있다.

```
/home/hpss-> get -R dir_name
```

□ ls

○ pftp_client와 다르게 ls 명령어에는 다음과 같은 다양한 옵션을 적용하여 필요한 정보를 보다 자세히 볼 수 있다

○ -U 옵션은 HPSS specific 정보를 볼 수 있다. 즉 COS 및 현제 저장되어 있는 물리적 위치 등을 알 수 있다.

Purpose:	List HPSS nodes
Aliases:	LIST, DIR
Command Format:	{ls | l[ist] | dir} [options] [path ...]
Options:	-a : list all entries, including "hidden" files whose names begin with "." -c : use time of last modification for sorting (deferred) -d : if file is a dir, list its name instead of its contents -l : long list format -p : put a slash after each name if the file is a directory (deferred) -r : reverse alpha or age sort order, as appropriate (deferred) -s : display size as well as name if -1 (numeral 1) option used -u : use time of last access for sorting instead of last modification (deferred) -x : multicolumn output format, with entries sorted across page (deferred) -A : print annotation info -C : multicolumn output format, with entries sorted down the columns (deferred) -F : puts a / after directory filenames, or * if executable (deferred) -H : print headings on long listings -O : print unordered "-l" or "-1" format listings -P : print one-line with position/volume info -R : recursively list directories -T : "type" where type is one of w,r,c or m. This allows the user to specify which HPSS time value is displayed when one of the "long list" options (e.g. "ls -l") is used. The letters stand for: w - last write time (default) r - last read time c - file creation time m - time of last metadata update -U : print HPSS-specific information -V : print volume for 1st tape level -X : print extended volume info (for all levels) -1 : (numeral 1) forces one name-per-line list
U s a g e Notes:	List options are available to cause the output to be formatted in a variety of ways. They may be specified individually (e.g., "-C") or as a string (e.g., "-1R")
Example:	ls -l

□ du

○ 이 명령어는 모든 파일의 바이트 수를 나타낸다(유닉스의 du명령어와 유사)

Purpose:	DU gives the number of bytes contained in all files and, recursively, directories within each specified directory or file name. If name is missing, "." is used.
Command Format:	du [-?] [-a] [-b] [-e] [-k] [-s] [-w] [path ...]
Options:	-a : if specified, causes an entry to be printed for each file. -b : if specified, only counts files written since the specified date. The date is of the form "yyyy/mm/dd". -e : if specified, only counts files written before or on the specified date. The date is of the form "yyyy/mm/dd". -k : if specified, prints values in kilobytes (1k = 1024). Default is 512 byte blocks. -s : if specified, causes only the grand total to be printed. -w : if specified, only counts files written within the last 'n' days
Usage Notes:	If neither -s or -a is specified, an entry is printed for each directory. An error occurs if both -w and (-b or -e) is specified in such a way that it is impossible for any files to qualify
Example:	du -k

□ 기타 명령어 - quit, del, help

○ hsi 종료할 때는 quit 명령어를 이용한다.

```
/home/hpss-> quit
```

○ HPSS에 전송된 파일을 삭제할 경우 del 명령어를 사용한다.

```
/home/hpss-> del file_name
```

○ 여러 개의 파일을 한꺼번에 삭제할 경우 pftp_client와 같이 별도의 명령어(mdel)가 있는 것이 아니고 put과 와일드 카드(?, * 등)를 이용 한다.

```
/home/hpss-> del *.c
```

○ /home/hpss-> 상태에서 사용할 수 있는 모든 명령어를 보려면 help를 사용한다.

```
/home/hpss-> help
```

□ 보다 자세한 명령어 사용법

○ 다음 사이트를 참조한다.(http://www.sdsc.edu/Storage/hsi/Doc/ch8.html)

3.4.3 배치작업에서의 HSI 사용

□ 개요

○ 배치작업에서의 HPSS 사용은 배치 스크립트 내에 간단한 코드를 삽입하는 것으로 간편하게 이용할 수 있다.

○ 주요 코드는 다음과 같다

```
hsi put filename          또는
hsi get filename
```

○ 배치작업에서의 HPSS 사용을 위한 환경설정으로는 .hsirc 및 keytab 파일을 만드는 것이다. 만드는 방법 및 설정방법은 < (다) 로그인 환경(자동로그인), 18page >을 참조한다.

3.5 Client API 사용법

3.5.1개요

Client API는 HPSS에서 제공되는 특수한 기능들, 예를 들어 storage/access hints passed on file creation, parallel data transfer, migration, and purge 등을 POSIX application 프로그래머들에게 제공하는 기능을 한다. 즉 자신이 프로그램 하는 코드 속에 HPSS에서 제공하는 함수들을 삽입하여 컴파일 후 실행시 데이터의 백업 및 retrieve의 기능을 일괄적으로 처리할 수 있다. 제공되는 API는 AIX용으로만 개발되어 있으며(현재 IBM p690, IBM p630에서만 서비스 가능) C 코드로 되어 있다.

Client API는 제공되는 주요 part는 다음과 같다

- File Open/Creation and Close Operations
- File Data Access Operations
- Fileset/Junction Creation and Deletion Operations
- File Attribute Operations
- File Name Operations
- Directory Creation and Deletion Operations
- Directory Access Operations
- Working Directory Operations

- Client API Control Operations
- DCE Login Context Routines
- Bitfile Server Statistic Operations

3.5.2 Building Processing

□ Makefile

다음은 “example1.c" 파일을 컴파일 하기 위한 Makefile이다

```
CC              = xlc_r7                                          #①
COMPFLAGS       = -g -DNO_DCE                                     #②
INCLUDE_PATH= -I. -I/system/super/hpssadm/nondce_hpss/include#③
CFLAGS          = $(INCLUDE_PATH) $(COMPFLAGS)

.c.o:;          @echo "Compiling $<"...
                @$(CC) $(CFLAGS) -c $<

LIBS            = -L/system/super/hpssadm/nondce_hpss/lib \  #④
                  -lhpss \
                  -lhpssapi \
                  -lm

PROG            = example1

all:            $(PROG)

clean:;         /bin/rm -f *.o $(PROG)

example1:       example1.o
                @echo "..... $@"...
                @$(CC) $@.o -o $@ $(CFLAGS) $(LIBS)
```

위의 코드를 “Makefile”이란 이름으로 저장한 다음 사용자 홈 디렉토리로 옮긴다. 주의할 점은 ①에서 컴파일러로 반드시 “xlc_r7”을 사용한다. ”xlc_r4“나 “xlc” 또는 ”cc_r4“, ”cc“등은 컴파일 에러를 발생시킨다.

컴파일러 flags 로 ②를 사용한다. ③, ④과 같이 헤더파일과 라이브러리 파일의 경로를 지정한다.

<주의>
위의 Makefile을 보면 “＼”가 보이는데 이것은 윈도우에서 “₩”로 나타난다. 그러므로 유닉스 홈 디렉토리로 옮기거나 또는 vi 에디터로 그냥 편집할 경우 주의를 기울여야 한다. 그리고 윈도우상에서 편집기로 위의 파일을 만들어서 유닉스로 옮길 경우 반드시 유닉스 상에서 위의 파일을 열어봐서 쓰레기 값이 들어가지 않았는지 확인한다.

□ 헤더 파일

○ 모든 프로그램에 다음 헤더 파일을 필수적으로 포함시킨다.

```
#inculde "hpss_api.h"
```

○ 64-bit 수학 라이브러리를 사용할 경우 다음의 헤더파일을 포함 시킨다

```
#inculde "u_signed64.h"
```

3.5.3 Basic Data Types and structures

□ Data Types

○ Client API에서 사용하는 기본 데이터 타입

- unsigned16	unsigned short integer
- unsigned32	32-bit unsigned integer
- signed32	32-bit signed integer
- u_signed64	64-bit unsigned integer
- hpssoid_t	HPSS object identifier type

○ 기타 일반적인 데이터 타입은 32-bit POSIX 데이터 타입을 사용한다.

□ Data Structure

HPSS Client API 전용 구조체로 다음 3가지가 있다.

○ Class of Service Priorities Structure - hpss_cos_priorities_t

```
typedef struct hpss_cos_priorities {
                    unsigned32 COSIdPriority
                    unsigned32 COSNamePriority
                    unsigned32 OptimumAccessSizePriority
                    unsigned32 MinFileSizePriority
                    unsigned32 MaxFileSizePriority
                    unsigned32 AccessFrequencyPriority
                    unsigned32 TransferRatePriority
                    unsigned32 AvgLatencyPriority
                    unsigned32 WriteOpsPriority
                    unsigned32 ReadOpsPriority
                    unsigned32 StageCodePriority
                    unsigned32 StripeWidthPriority
                    unsigned32 StripeLengthPriority
          } hpss_cos_priorities_t
'*Priority' Settings available for priority values:
            NO_PRIORITY   LOW PRIORITY   DESIRED_PRIORITY
          HIGHLY_DESIRED_PRIORITY   REQUIRED_PRIORITY
```

○ File Creation Hint Structure - hpss_cos_hints_t

```
typedef struct hpss_cos_hints {
                    unsigned32 COSId
                    char       COSName[HPSS_MAX_OBJECT_NAME];
                    u_signed64 OptimumAccessSize
                    u_signed64 MinFileSize
                    u_signed64 MaxFileSize
                    unsigned32 AccessFrequency
                    unsigned32 TransferRate
                    unsigned32 AvgLatency
                    unsigned32 WriteOps
                    unsigned32 ReadOps
                    unsigned32 StageCode
                    unsigned32 StripeWidth
                    u_signed64 StripeLength
                } hpss_cos_hints_t
```

COSId	Class of service id
COSName	Class of service name
OptimumAccessSize	Client-access block size in bytes that yields maximum data transfer rate
MinFileSize	Minimum file size in bytes
MaxFileSize	Maximum file size in bytes
AccessFrequency	Expected rate of access, which can be one of the following: FREQ_HOURLY FREQ_DAILY FREQ_WEEKLY FREQ_MONTHLY FREQ_ARCHIVE
TransferRate	Approximate file transfer rate in KB per second
AvgLatency	Average latency
WriteOps	Bitmask specifying valid write operations, which can be one or more of the following: HPSS_OP_RANDOM HPSS_OP_WRITE HPSS_OP_PARALLEL HPSS_OP_WRITE_MANY HPSS_OP_SEQUENTIAL HPSS_OP_APPEND
ReadOps	Bitmask specifying the valid read operations for the bitfile, which can be one or more of the following values: HPSS_OP_RANDOM HPSS_OP_SEQUENTIAL HPSS_OP_PARALLEL HPSS_OP_READ
StageCode	Staging behavior desired. One of the following values can be used: COS_STAGE_NO_STAGE COS_STAGE_ON_OPEN COS_STAGE_ON_OPEN_ASYNCH COS_STAGE_ON_OPEN_BACKGROUND
StripeWidth	The stripe width of the COS
StripeLength	The stripe length of the COS

○ HPSS Directory Entry Structure - hpss_dirent_t

```
typedef struct hpss_dirent {
                    unsigned32          d_offset
                    ns_ObjHandle_t      d_handle
                    unsigned16          d_reclen
                    unsigned16          d_namelen
                    char                d_name[HPSS_MAX_FILE_NAME];
          } hpss_dirent_t
```

d_offset Offset of next directory entry
d_handle Handle value to the Name Server of the directory
d_reclen Record length of the directory
d_namelen Number of characters in the directory entry name
d_name Name of the directory entry

3.5.4 64-bit Math Library

□ 자주 쓰이는 64-bit 함수 및 매크로

○ add64m : Add two 64 bit unsigned integers
  · u_signed64 add64m(u_signed64, u_signed64)
○ cast64m : Cast 32 bit number into 64 bit value
  · u_signed64 cast64m (unsigned32)
○ eq64m : Test two 64 bit numbers
  · int eq64m (u_signed64, u_signed64)
    Return value: 1 if equal, 0 if not equal
○ high32m : Return high 32 bit part of 64 bit value
  · unsigned32 high32m (u_signed64)
○ low32m : Return low 32 bit part of 64 bit value
  · unsigned32 low32m (u_signed64)
○ lt64m : Compares two 64 bit unsigned numbers
  · int lt64m (u_signed64, u_signed64)
    Return value: 1 if 1st value is less than 2nd arguments, 0 otherwise
○ sub64m : Subtract two 64 bit values, 1st arg - 2nd arg
  · u_signed64 sub64m (u_signed64, u_signed64)
○ u64_to_decchar : Convert unsigned 64 bit value to string (base 10)

```
#include "u_signed64.h"
int  u64_to_decchar (u_signed64  *Value64,   /* IN  */
                     char        *String)    /* OUT */
```

Value64 Unsigned 64-bit value to convert to a character string
String Pointer to buffer that will contain the converted value. *String* must be large enough to contain up to 20 characters

Return value:
A value of 0 is always returned. Otherwise a negative value of **ERANGE** or **EINVAL** will be returned indicating the error

◯ decchar_to_u64 : Convert string (base 10) to 64-bit unsigned value

```
#include "u_signed64.h"
int  decchar_to_u64 (char        *String,   /* IN  */
                     u_signed64  *Value64,  /* OUT */
                     int         Length)    /* IN  */
```

String Point to buffer with 64-bit string value to convert
Value64 Pointer to 64-bit value to receive the converted value
Length Length of *String* value, or zero (*String* must be NULL terminated)

Return value:
Upon success, a value of 0 is returned. Otherwise a negative value of **ERANGE** or **EINVAL** will be returned indicating the error

3.5.5 Error Code

□ 일반적인 에러 처리

◯ API 루틴들은 에러 조건에서 negative number를 리턴 한다.
- 에러 코드의 절대 값은 표준 POSIX/errno 에러 코드와 동일하다(예를 들어 EPERM, ENOENT, EIO 등)

◯ 대부분의 함수는 성공 시 zero(0)을 리턴 한다.

◯ 일부 함수들은 positive value를 리턴 한다.
- 지시자(예를 들어 hpss_Open()으로부터의 파일 디스크립터)
- 바이트 수(예를 들어 hpss_Read()에서 읽은 바이트)
- 기타 값(예를들어 hpss_Umask())

◯ 2가지 예외 함수 : hpss_SetLoginContext() & hpss_PurgeLoginContext()
- POSIX mapped error를 나타낼 때 negative number

- 인증에 실패 시 positive DCE error code 리턴

□ 인증

○ “hpss_SetCombo" 함수 사용

예를 들어 HPSS 계정 및 패스워드가 각각 "hpss"라면 다음과 같이 프로그램에 쓴다.

- hpss_SetCpmbo("hpss", "hpss");

```
#include "hpsscomm.h"
 signed32 hpss_SetCombo (char *DCEPrincipalName,  /* IN */
                         char *Password)          /* IN */

        DCEPrincipalName      String containing user's DCE principal name
        Password              String containing user's password (no
                              encryption)

Return value:
Upon success, a value of 0 is returned.  Otherwise a negative value indicating the
error will be returned
```

○ Kerboros 인증은 지원하지 않는다.

3.5.6 예제 코드

다음은 사용자가 빈번히 사용할 수 있는 예제 프로그램이다. 이 프로그램은 파일을 아큐멘트에서 받아서 HPSS 파일 시스템으로 writing 하는 기능을 가진다. 실제 프로그램에서는 생성된 파일을 아규멘트로 받기 보다는 프로그램 내에서 데이터가 생성되고 이를 자동적으로 스토리지에 writing하는 경우가 많으므로 사용자들은 이 예제 프로그램을 기반으로 사용자 프로그램의 특성에 맞게 변경할 필요가 있다.

다음은 소스에 대한 보다 자세한 코멘트는 “다. 프로그램 상세설명”을 참조한다.

```
/*
 * 프로그램 : ex1.c
 * 코드설명 : 로컬 파일을 HPSS filesystem으로 복사(writing)하는 프로그램
```

```
 * 가정 : 사용자는 HPSS계정을 기지고 있어야 한다.
 */

#include <stdio.h>
#include <errno.h>
#include <fcntl.h>
#include <sys/time.h>
#include <sys/stat.h>

/*
 * 다음 2개의 헤더 파일은 HPSS API를 사용하기 위해 필수적으로 삽입된다.
 */
#include "hpss_api.h"
#include "u_signed64.h"

/*
 * 다음은 전송을 위한 버퍼와 파일 퍼미션을 정의한다.
 * 버퍼 크기는 조건에 따라 임으로 늘릴 수 있다.
 * 퍼미션 또한 조건에 따라 임으로 변경할 수 있다. 이 퍼미션 플랙은
hpss_Open함수를
 * 사용할 때 이용된다. 여기서는 -rw-r--r--로 정의하였다.
 */

#define BUFSIZE (128*1024*1024) /* "다. 프로그램 상세 설명" 참조 */
#define PERMS 0640

/*
 * COS값을 리턴하는 함수 이다
 */
int get_cos(int);

/*
 * 파일의 전송시간, 전송률 등을 리포팅 해줄 수 있는 함수이다.
```

```
 * 옵션으로 사용할 수 있다.
 */
double double_time();


/*
 * 매인 함수의 시작
 */

int main(int argc, char *argv[])
{
        int                     fd, hfd, rc, eof, len, i, sum_byte;
        char                    *buf;
/*
 * File Creation Hint Structure
 */
        hpss_cos_hints_t hints_in;
/*
 *Class of Service Priorities Structure
 */
        hpss_cos_priorities_t   hints_pri;
        struct stat             buff;
        int                     getcos;
        double                  start, elapsed;
        double                  sum_time;

/*
 * Make sure use passed in 2 arguments
 * 이 프로그램의 사용법은
 *      %./ex1 localfile
 * 으로 2개의 argument가 사용된다.
 */
        if ( argc != 2 )
```

```
        {
            fprintf(stderr, "Usage: %s localfile₩n", argv[0]);
            exit(1);
        }
/*
 * HPSS로 전송하고자 하는 local file을 open한다.
 */
        fd = open(argv[1], O_RDONLY);
        if ( fd < 0 )
        {
            perror(argv[1]);
            exit(1);
        }
/*
 * 다음은 그 local file의 크기를 알아내기 위해 stat함수를 사용하였다.
 * 왜 파일의 크기를 알아야만 하는지는 “다. 프로그램 상세 설명”을 참조 한
다.
 * stat 함수의 사용법은 UNIX 프로그래밍 서적을 참조 한다.
 */
        stat(argv[1], &buff);
/*
 *local file 크기에 따라 COS값을 리턴한다.
 /*
        getcos = get_cos(buff.st_size);
/*
 * 다음은 옵션이며 주석처리해도 상관없다.
 */
        printf("The COS of %s is %d₩n", argv[1], getcos);
/*
 * Initialize the hpss hints structures
 * 다음 초기화들은 필수적으로 해줘야 한다.
 */
        memset(&hints_in, 0, sizeof(hints_in));
```

```
        memset(&hints_pri, 0, sizeof(hints_pri));
        hints_in.COSId = getcos;
        hints_pri.COSIdPriority = REQUIRED_PRIORITY;

/* 인증
 * 다음은 HPSS로  접근할 수 있는 로그인 함수이다
 * HPSS 계정 아이디가 “hpss"이고 암호도 ”hpss"인 것으로 가정한다.
 */
        hpss_SetCombo("hpss", "hpss");

/*
 * Allocate memory for transfers
 * HPSS filesystem으로 local file를 전송하기 전에 임시적으로 저장될 버퍼 메모리를
 * 할당한다. 여기에 임시적으로 저장 되었다가 hpss_Write함수에 의해 HPSS로 writing
 * 된다.
 */
        if ((buf = (char *)malloc(BUFSIZE)) == NULL)
        {
            fprintf(stderr, "Could not allocate memory₩n");
            exit(1);
        }
/*
 * Create the HPSS file with specific HPSS hints
 * HPSS filesystem에 writing될 파일을 open한다.
 */
        hfd = hpss_Open(argv[1], O_WRONLY | O_CREAT | O_TRUNC,
                        PERMS, &hints_in, &hints_pri, NULL);
        if (hfd < 0)
        {
            fprintf(stderr, "Could not open/create: ");
            fprintf(stderr, "hpss_Open: %s₩n", strerror(-hfd));
```

```
            exit(1);
        }

/*
 * Loop for copying the data
 * 다음은 local file이 EOF가 될 때까지 읽어서 버퍼에 저장했다가 hpss_Write 함수에
 * 의해 hpss_Open으로 create된 파일에 쓰여 진다.
 * 파일이 쓰여 지는 동안에 double_time 함수에 의해 전송 시간이 check된다.
 */
        sum_byte = 0;
        sum_time = 0;
        for ( eof = 0 ; !eof ; )
        {
/*
 * Try to read in BUFSIZE worth of data
 */
            for ( len = 0 ; !eof && len < BUFSIZE ; len += rc )
            {
                rc = read(fd, buf+len, BUFSIZE-len);
                if ( rc <= 0 ) eof = 1;
            }
/*
 * Now write the buffer contents to the HPSS file
 */
            if ( len > 0 )
            {

/*
 * start the timer
 */
                start = double_time();
                rc = hpss_Write(hfd, buf, len);
```

```
                if ( rc != len )
                {
                    fprintf(stderr, "hpss_Write: %s₩n", strerror(-rc));
                    exit(0);
                }
/*
 * stop the timer
 */
                elapsed = double_time() - start;
                sum_byte += len;
                sum_time += elapsed;
/*
 * 전송 상황을 출력한다.
 */
                printf("%d Bytes in %.3f sec -> %.3f MB/sec₩n", len,
elapsed, (double)len/1000.0/1000.0/elapsed);
            }
        }

/*
 * Report the amount of data transferred
 */
        printf("₩n");
        printf("Finished Transferring %s : %d Bytes in %.3f sec -> %.3f
MB/sec₩n", argv[1], sum_byte, sum_time, sum_byte/sum_time/1000000.0);
        printf("₩n");



/*
 * Close out the local and HPSS file
 */
        close(fd);
```

```
        hpss_Close(hfd);
        return(0);
}

int get_cos(int size)
{
/*
 * 다음 3라인은 COS(Class of Service)에 대한 상수 정의 부분이다, 이부분도
프로그램
 * 코딩시 필수적으로 입력되어야 한다. "다. 프로그램 상세설명" 참조
/*
        const int cos113 = 2 << 29;
        const int cos112 = 2 << 25;
        const int cos111 = 2 << 19;

        if(size > cos113) return 113;
        else if(size > cos112) return 112;
        else if(size > cos111) return 111;
        else return 110;
}

double double_time()
{
        struct timeval tv;
        gettimeofday(&tv, NULL);
        return((double)(tv.tv_sec) + (double)(tv.tv_usec)/1000000.0);
}
```

□ 컴파일

HPSS API를 사용한 소스 코드는 반드시 위에서 설명한 Makefile을 만들어서 컴파일 해야 한다. 위의 예제 소스 코드를 컴파일 하기 위해서는 “2. Building process”에서 설명한 Makefile에서 example1 대신에 ex1을 example1.o 대신에 ex1.o로 바꾸어서 저장한 다음 명령어를 수행한다.

```
% make
Compiling ex1.c...
..... ex1...
Target "all" is up to date.
```

□ 실행

다음은 1,073,741,824 바이트의 파일(파일이름 : 10m)을 전송시키는 예를 보여준다.

```
nobela:/system/super/hpssadm/test_api/examples1>./ex1 10m
The COS of 10m is 112
134217728 Bytes in 2.264 sec -> 59.273 MB/sec
134217728 Bytes in 1.954 sec -> 68.677 MB/sec
134217728 Bytes in 1.921 sec -> 69.877 MB/sec
134217728 Bytes in 1.926 sec -> 69.700 MB/sec
134217728 Bytes in 1.953 sec -> 68.716 MB/sec
134217728 Bytes in 1.930 sec -> 69.533 MB/sec
134217728 Bytes in 1.928 sec -> 69.601 MB/sec
134217728 Bytes in 1.921 sec -> 69.859 MB/sec

Finished Transferring 10m : 1073741824 Bytes in 15.798 sec -> 67.966 MB/sec

nobela:/system/super/hpssadm/test_api/examples1>
```

1,073,741,824 바이트의 파일을 writing하는데 15.798초에 67.966 MB/sec의 전송률을 나타낸다. 이러한 전송시간과 전송률은 시스템 환경에 따라 많이 차이가 난다. 예를 들어 사용자가 집중적으로 사용한다든가 또는 네트워크 장

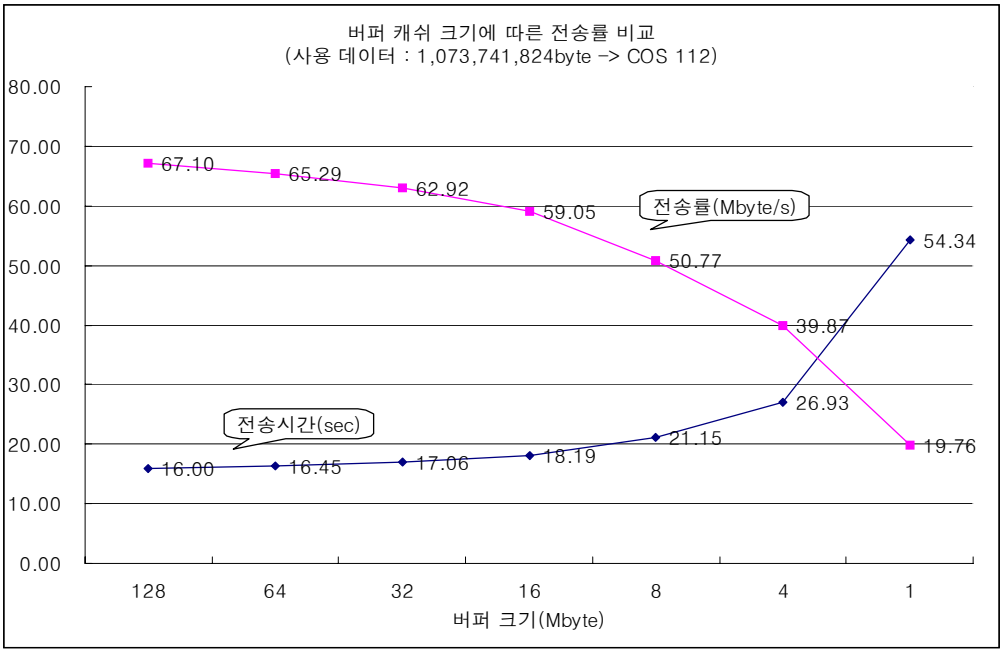
애등이 그런 이유이다. 테스트용 파일인 “10m”은 테스트용으로 임의 제작된 것으로 버퍼블록 사이즈(n*1024*1024)에 맞추어서 만든 것이다. 그렇기 때문에 최적의 전송률을 나타낼 수 있었으리라 생각된다. 실제 상황에서는 이보다 낮은 전송률을 나타낼 것이다.

□ 프로그램 상세 설명

○ 버퍼 사이즈

버퍼 사이즈는 전송속도에 큰 영향을 미친다, 아래 그림은 각 버퍼 사이즈에 따른 전송시간과 전송률을 보여주고 있다. 버퍼 사이즈가 128MB이상일 경우 memory allocation error가 발생한다. 버퍼 사이즈는 다음 공식으로 할당한다.

n*1024*1024



○ COS할당

현재 KISTI HPSS 시스템의 COS는 다음과 같이 정의되어 있다(COS에 대한 자세한 설명은“HPSS 사용자 가이드”를 참조).

COS ID	min. byte	max. byte
110	0	1,048,576
111	1,048,577	67,108,864
112	67,108,865	1,073,741,824
113	1,073,741,825	unlimited

즉, writing할 데이터의 파일 크기에 따른 COS를 “COS Hint Structure"의 멤버 변수 값으로 할당해야 한다. 위의 예제 프로그램에서는 다음과 같이 코딩되어 있다

```
struct stat                buff;
hpss_cos_hints_t           hints_in;
 ...
stat(argv[1], &buff)
getcos = get_cos(buff.st_size);
        ...
hints_in.COSId = getcos;
```