hp e3000

programming and posix

programming and posix

presented by Mark Bixby
mark_bixby@hp.com

Solution Symposium 2001

contents

- Getting Started
- The Hierarchical File System (HFS)
- Files and Directories - A Review
- A Simple Program and a CGI Program
- Creating and Linking with Libraries
- POSIX API Topics
- POSIX Shell Topics
- Additional Programming Topics

getting started

- Logon:

```
:hello <user>.<account>
```

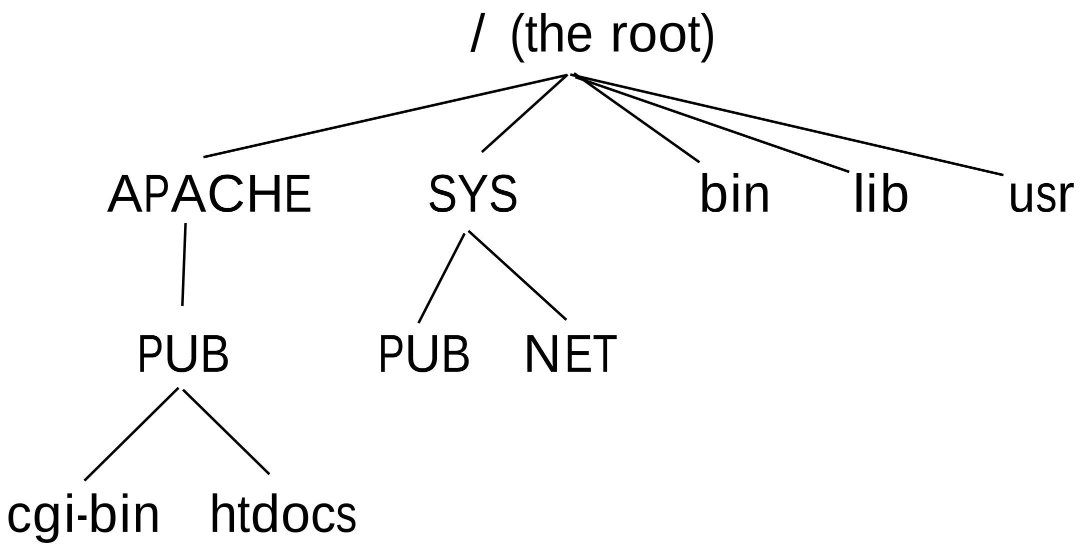
- Enter the POSIX shell:

```
:xeq sh.hpbin.sys -L
```

- Exit the POSIX shell:

```
shell/iX> exit
```

the hierarchical file system (hfs)



/ (the root)

APACHE SYS bin lib usr

PUB PUB NET

cgi-bin htdocs

Absolute path: /APACHE/PUB/cgi-bin/hwcgi

Relative path: ./hwcgi

working with files - a review

- Naming a file
  - 16 character limit if below an MPE account or group
  - 256 character limit otherwise
- File Types - bytestream vs. fixed record
- Creating and listing files - cat >, vi, ls
- Viewing and printing a file - more, cat, lp
- Copying, renaming, and removing files - cp, mv, rm
- Displaying and changing a file’s permissions and ownership - chmod, chown, chgrp

organizing files with directories - a review

- Displaying your current directory - pwd
- Absolute and relative pathnames
  - /an/absolute/pathname
  - a/relative/pathname
  - ./another/relative/pathname
  - ../upward/relative/pathname
- Changing to a different directory - cd
- Creating a new directory - mkdir
- Removing a directory - rmdir
- Recursively scan directories - find

file and directory security

- each object is owned by a POSIX user (UID) and a POSIX group (GID)
  - POSIX UID maps to an MPE USER.ACCOUNT
  - POSIX GID maps to an MPE ACCOUNT
- Three independent security classifications:
  - Do you match the object’s user?
  - Else do you match the object’s group?
  - Else then you’re “other”
- Three independent types of access per classification:
  - read (r)
  - write (w)
  - execute (x)

permission mode bits

User	Group	Other
rwx	rwx	rwx

- Specified in chmod command symbolically or as 3 octal digits:
  - `chmod u=rwx,g=rx,o=x file`
  - equivalent to `chmod 751 file`
- The umask command and function specifies a mask of permission modes to be disabled when files are created
  - `umask 007` denies all access to “other”
  - remains in effect until another umask or logoff

file security example

```
shell/iX> chmod 751 file
shell/iX> ls -l file
-rwxr-x--x 1 MANAGER.SYS  SYS   0 Jan  3 13:29 file

shell/iX> chmod 644 file
shell/iX> ls -l file
-rw-r--r-- 1 MANAGER.SYS  SYS   0 Jan  3 13:29 file
```

vi editor

- the only bytestream file editor supplied by CSY
- hated by many, but standard on all Unixes
- command mode vs. data entry mode
  - starts in command mode
  - almost every character is a command
  - some commands toggle to data entry mode
  - ESC terminates data entry mode

vi command quick reference

- a - append to the right of cursor (data entry mode)
- i - insert to the left of cursor (data entry mode)
- o - add a new line below cursor (data entry mode)
- O - add a new line above cursor (data entry mode)
- dd - delete the current line
- x - delete the current character
- r - replace current character with next typed character
- cw - change current word (data entry mode)
- dw - delete current word
- . - repeat last modification command

vi command quick reference (cont.)

- space - move cursor right
- backspace - move cursor left
- return - move cursor down
- hyphen - move cursor up
- `/string` return - search forward
- `:1,$s/foo/bar/g` - replace all foo with bar every line
- `:wq` - save and quit

compiling - gcc vs. c89

- Use gcc if:
  - you’re porting an open-source application
  - you want to write portable code that will run on other platforms
  - support contracts available from http://www.gccsupport.com
- Use HP c89 if:
  - you’re calling many MPE intrinsics
    - `#pragma instrinsic`
  - you need long pointer support
  - you want to use an HP-supported compiler

hp e3000

programming and posix

a simple program and a cgi program

- A Simple Program
  - Create the file
  - Compile and link
  - Run it

- A CGI Program
  - Create the file
  - Compile and link
  - Test it
  - Run it from a web browser

a simple program - 1

- Create the source file hw.c:

```
#include <stdio.h>   /* printf() */

main()
{
   printf("hello world\n");
}
```

a simple program - 2

- Compile and link the source file:

```
shell/iX> gcc -o hw -D_POSIX_SOURCE hw.c
```

  - `-o` specifies the output file NMPRG
  - `-D` defines the symbol `_POSIX_SOURCE` which is required for all POSIX programming

- Run the program:

```
shell/iX> hw
hello world
```

a cgi program - 1

- Edit the source file:

```
shell/iX> cp hw.c hwcgi.c

shell/iX> vi hwcgi.c
#define _POSIX_SOURCE /* instead of -D */
#include <stdio.h>

 main()
 {
    printf("Content-type: text/plain\n\n");
    printf("hello world\n");
 }
```

- Compile and link the program:

```
shell/iX> gcc -o hwcgi  hwcgi.c
```

a cgi program - 2

- Test the CGI program:

```
shell/iX> echo foo | hwcgi | cat
Content-type: text/plain
hello world
```

- Copy CGI program to cgi-bin directory:

```
shell/iX> cp hwcgi /APACHE/PUB/cgi-bin
```

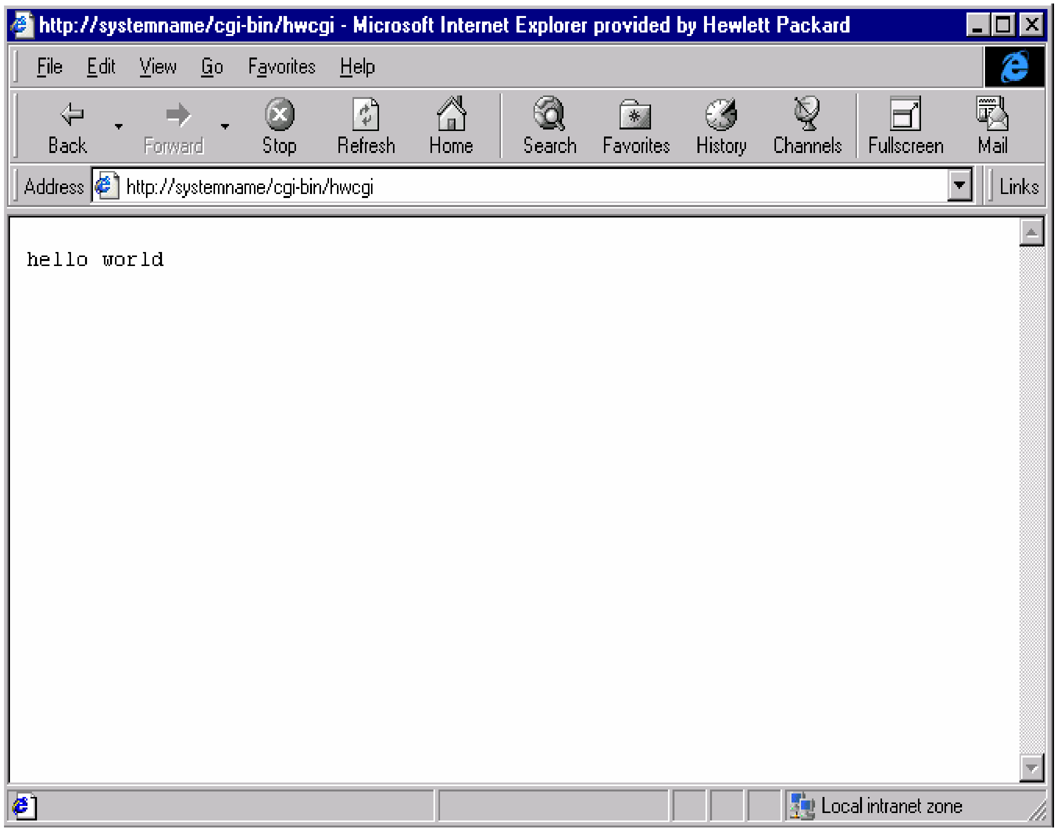
- Point browser at:

```
http://systemname/cgi-bin/hwcgi
```

hp e3000

programming and posix

a cgi program - 3


http://systemname/cgi-bin/hwcgi - Microsoft Internet Explorer provided by Hewlett Packard
File Edit View Go Favorites Help
Back
Forward
Stop
Refresh
Home
Search
Favorites
History
Channels
Fullscreen
Mail
Address
http://systemname/cgi-bin/hwcgi
Links
Local intranet zone

```
hello world
```

creating an nmrl archive library - 1

- Write new helloworld() function in helloworld.c:

```
#define _POSIX_SOURCE
#include <stdio.h>

helloworld()
{
        printf("hello world\n");
}

shell/iX> gcc -c helloworld.c
```

  - `-c` generates an NMOBJ instead of NMPRG

- Create the NMRL archive library:

```
shell/iX> ar -rv libhw.a helloworld.o
```

  - r - replace or add the object to the library
  - v - verbose

creating an nmrl archive library - 2

- Have our main program in hwcgimain.c call an external:

```
#include <stdio.h>
extern void helloworld(void);
main()
{
	printf("Content-type: text/plain\n\n");
	helloworld();
}

shell/iX> gcc -c -D_POSIX_SOURCE hwcgimain.c
```

- Link the program:

```
shell/iX> gcc -o hwcgi hwcgimain.o -L. -lhw
   or
shell/iX> ld -o hwcgi hwcgimain.o -L. -lhw
```

  - `-L.` specifies library search directory (. is CWD)
  - `-lhw` refers to libhw.a

creating an nmxl shared library

- Create the NMXL shared library:

```
shell/iX> gcc -Wl,-b -o libhw.sl helloworld.o
or
shell/iX> ld -b -o libhw.sl helloworld.c
```

  - `-Wl` - pass parameters to linker
  - `-b` - create an NMXL shared library

- Link with the shared library:

```
shell/iX> gcc -o hwcgi hwcgimain.o \
              -Wl,-ashared -L. -lhw
```

  - `-ashared` - prefer NMXLs instead of NMRLs

linking with system libraries

- libc is included in link by default

```
shell/iX> gcc -o hwcgi hwcgi.c
```

- System libraries located in /lib and /usr/lib
  - libc, libsvipc are in /lib
  - libsocket is in /usr/lib
- System libraries exist in both archive and shared form (as of MPE 6.0). During link,
  - NMRL archive library (.a suffix) merged into program
  - NMXL shared library (.sl suffix) is NOT merged

linking with libraries - syntax

- `-lfoo` means link with library libfoo.a
  - `-lc` is included in link by default
- `-Lpath` tells where library is located
  - `-L/lib -L/usr/lib` is included in link by default

- Link with libsvipc archive library

```
shell/iX> gcc -o hwcgi hwcgi.c -lsvipc
```

- Link with libsvipc shared library

```
shell/iX> gcc -o hwcgi hwcgi.c -Wl,-ashared
  -lsvipc
```

  - `-Wl,-ashared` specifies shared library preference

gcc vs. ld for linking

- ld can create NMPRGs and NMXLs (shared libraries)
- but use gcc instead so that required gcc compiler libraries will be linked in

make utility

- Rebuilds only those components which need rebuilding based on which dependent files have newer timestamps
- A simple Makefile:

```
all: hwcgi

hwcgi: hwcgimain.o libhw.a
        $(CC) -o $@ hwcgimain.o -L. -lhw

libhw.a: helloworld.o
        $(AR) $(ARFLAGS) $@ $?
```

- `make` will compile and link everything that needs updating
- `make -n` to display commands without execution

posix api topics

- Program Parameters
- Environment Variables
- File Management
- Process Management
- User and Group Management
- InterProcess Communication
- Sockets
- Signals
- Error handling

program parameters

- `int argc` - number of parameters
- `char **argv` - pointer to list of parameter pointers
  - `argv[0]` - name of program as invoked by user

```
int main(int argc, char **argv) {

int i;
/* print all parameters */
for (i=0; i < argc; i++) {
  printf(“argv[%d] = %s\n”,i,argv[i]);
}
}
```

environment variables

- special string variables copied from parent to child when new processes are created
  - the POSIX shell will only copy exported variables
    - `foo=bar; export foo`
    - `export foo=bar`
  - static copies of CI variables are exported by the shell
- the environment is a process-local structure; your parent and any already running children won't see any environment changes that you make

environment variables - reading

```
#include <stdlib.h>
#include <stdio.h>

int main() {
char *foo_value;

if ((foo_value = getenv("FOO")) == NULL) {
 printf("FOO not found!\n"); exit(1);
}

printf("FOO=%s\n",foo_value);
}
```

environment variables - writing

```
#include <stdlib.h>
#include <stdio.h>

int main() {

if (putenv("BAR=456")) {
  printf("putenv failed"); exit(2);
}

system("echo $BAR"); /* executes in a child shell */
}
```

file management

- POSIX file descriptors instead of MPE file numbers
  - a 32-bit integer field
- just a stream of bytes - no record boundaries!
- certain reserved file descriptors are always opened when a process is created:
  - 0 - stdin (program input)
  - 1 - stdout (program output)
  - 2 - stderr (program error message output)

file management - open()/close()

- `int open (const char *path, int oflag, int modes);`
  - returns -1 for error, else returns file descriptor
  - ONE of O_RDONLY, O_WRONLY, O_RDWR
  - O_APPEND - file offset set to EOF prior to writes
  - O_CREATE - opt. permission modes parm is req'd
  - O_EXCL - exclusive access
  - can function as a semaphore by specifying both O_CREATE and O_EXCL which will return an error if the file already exists, else will create the file
- `int close (int fildes);`
  - can also be used on socket descriptors

file management - read()/write()

- `ssize_t read (int fildes, void *buffer, size_t nbyte);`
  - returns number of bytes actually read or -1 if error
  - can also be used on socket descriptors

- `ssize_t write (int fildes, const void *buffer, size_t nbyte);`
  - returns number of bytes actually written or -1 if error
  - can also be used on socket descriptors

file management - lseek()

- `off_t lseek (int fildes, off_t offset, int whence);`
  - changes the current file position
  - returns the new file offset or -1 if error
  - offset - number of bytes
  - whence - how the offset is applied to the current position:
    - `SEEK_SET` - Set new offset to *`offset`*.
    - `SEEK_CUR` - Set new offset to *`offset`* plus the current offset.
    - `SEEK_END` - Set new offset to *`offset`* plus the current file size.

file management - stat()

- `int stat (const char *path, struct stat *buffer);`
- `int fstat (int fildes, struct stat *buffer);`
- `int lstat (const char *path, struct stat *buffer);`
    - reports on the symlink instead of the target file
- obtains file attributes
- some `struct stat` fields from <sys/stat.h>:
    - `st_mode` - permission modes
    - `st_uid`, `st_gid` - POSIX uid & gid of owner
    - `st_size` - file size in bytes
    - `st_atime`, `st_mtime`, `st_ctime` - accessed, modified, created timestamps

file management - dup()/dup2()

- duplicates file descriptors
- commonly used with fork()/exec() to create pipes
- `int dup (int fildes);`
- `int dup2(iint fildes, int fildes2);`
  - *`filedes2`* specifies the desired new descriptor number
  - commonly used to redirect stdin/stdout/stderr

file management - fcntl()

- `int fcntl(int *fildes, int cmd, ...);`
- Duplicate an existing file descriptor
- Get & set file descriptor flags
- Get & set file status flags
- Record locking
- sfcntl() must be used for socket descriptors

process management - fork() - 1

```
#include <unistd.h>

if ( (pid = fork()) < 0) {   /* error */
      perror("fork");
} else if (pid == 0) {       /* child */
      printf(“child: here\n”);
} else { /* if pid > 0 */    /* parent */
      printf(“parent: here\n”);
}
```

- clones the current process, creating an identical child with the exact same run-time state (open files, stack trace, local variable values, etc)

process management - fork() - 2

- Compile & link sample program

```
shell/iX> gcc -o forkt forkt.c
```

- Program & user must have PH capability
  - gcc link adds PH capability by default to program
- Run sample program

```
shell/iX> forkt
child: here
parent: here
```

process management - exec()

- 6 forms: execl(), execve(), execvp(), execv(), execve(), execvp()
- replaces the current process with a newly spawned one

```
if ( (pid = fork()) < 0)
     perrror("fork");
else if (pid == 0)      /* child */
{
     if (execl("/bin/echo",
               "echo", "child:", "hello",
"world", (char *) 0) < 0)
       perror("execl");
     printf("child: this never prints\n");
}
```

process management - execl()

- Compile & link sample program

```
shell/iX> gcc -o execlt execlt.c
```

- Run sample program

```
shell/iX> execlt
child: hello world
parent: exiting
```

- A child process on MPE will not survive the death of its parent; implement daemons via batch jobs instead

process management - getpid()/getppid()

- `int getpid (void)`
  - returns the POSIX PID of the calling process

- `pid_t getppid (void);`
  - returns the POSIX PID of the parent of the calling process

user management - getuid()/setuid()

- `uid_t getuid (void);`
  - returns the POSIX UID of the calling process
- `int setuid(uid_t` *`uid`*`);`
  - changes the POSIX UID of the calling process
  - requires GETPRIVMODE()
  - if you change to a UID in another MPE account, the POSIX GID will also be changed to match the new account

hp e3000

programming and posix

user management - getpwnam()/getpwuid()

- `struct passwd *getpwnam(const char *name);`
- `struct passwd *getpwuid(uid_t uid);`
- reads from virtual /etc/passwd user directory file
  - /etc/passwd does not exist on MPE
- selected `struct passwd` fields from /usr/include/pwd.h
  - `pw_name` - user name (USER.ACCOUNT)
  - `pw_uid` - POSIX UID
  - `pw_gid` - POSIX GID
  - `pw_dir` - initial working directory (MPE home group)
  - `pw_shell` - initial shell (/SYS/PUB/CI)

group management - getgid()/setgid()

- `gid_t getgid (void);`
  - returns the POSIX GID of the colling process

- `int setgid(gid_t *gid*);`
  - exists but isn't implemented
  - MPE forces your GID to correspond to the MPE account of your UID anyway

group management - getgrgid()/getgrnam()

- `struct group *getgrgid(gid_t gid);`
- `struct group *getgrnam(const char *name);`
- reads from virtual /etc/groups group directory file
  - /etc/groups does not exist on MPE
- selected `struct group` fields in /usr/include/grp.h:
  - `gr_name` - group name
  - `gr_gid` - POSIX GID
  - `gr_mem` - NULL-terminated list of pointers to individual group member names

interprocess communication (ipc)

- Pipes
  - pipe(fd[2])
  - popen() / pclose()
- FIFOs
  - mkfifo(pathname)

- Message queues (in libsvipc)
- Semaphores (in libsvipc)
- Shared memory (in libsvipc)

interprocess communication - pipes

- Pipes are easy to demonstrate in the shell:

```
shell/iX> who am i
STEVE,CGI.APACHE@SYSTEMNAME ldev5  TUE 1:04P
shell/iX> who am I | cut -f1 -d' '
STEVE,CGI.APACHE@SYSTEMNAME
```

- int pipe(int *filedes*[2]);
  - creates two file descriptors for pipe endpoints
    - *filedes*[0] for reading, *filedes*[1] for writing
  - pipe()/fork()/dup2()/exec() to do stdout|stdin piping between two processes
- popen()/pclose()
  - spawns shell pipe to execute a command
  - BEWARE of shell metacharacter security holes!!!

pipes the easy popen() way

```
#include <stdio.h>
int main() {

FILE *mypipe; char buf[256];

mypipe = popen("callci showtime", "r"); /* readable pipe */
while (fgets(buf, sizeof(buf), mypipe) != NULL)
  printf("pipe read = %s\n",buf); /* read until EOF */
pclose(mypipe);

mypipe = popen("/bin/sed -e 's/^/pipe write = /'", "w");
fputs("testing\n",mypipe); fputs("123\n",mypipe); /* write 2 lines */
pclose(mypipe);

}
```

pipes the hard pipe() way

```
#include <stdio.h>
#include <unistd.h>

int main() {
int request[2], response[2]; char buf[256];

pipe(request); pipe(response); /* create request & response pipes */

if (fork() > 0) { /* parent */
  close(request[0]); close(response[1]); /* close unneeded ends*/
  write(request[1], "123\n", 4); close(request[1]); /* write req*/
  buf[read(response[0],buf,sizeof(buf))] = 0; /* read response */
  printf("result = %s\n",buf);
} else { /* child */
  close(request[1]); close(response[0]); /* close unneeded ends*/
  dup2(request[0],0); dup2(response[1],1); /*redirect stdin&stdout*/
  execl("/bin/sed", "/bin/sed", "-e", "s/^/pipe = /", NULL);
}
}
```

sockets

- InterProcess communciation via socket address:
  - Internet (32-bit IPv4 address, port number)
  - Unix (local file name)
- Functions
  - socket() - create socket descriptor
  - connect() - connect to a remote socket
  - bind() - to use specific listening socket (i.e. port 80)
  - listen() - establish queue for incoming TCP conns
  - accept() - wait for a new TCP connection
  - read()/recv(), write()/send() - data transmission
  - close() - close a socket

sockets - server example

```
mysock = socket(AF_INET, SOCK_STREAM, 0);
bind(mysock, <address of port 80>);
listen(mysock, queuedepth);

…begin main loop…
remotesock = accept(mysock, <remote address>);
read request with read() or recv()
write response with write() or send()
close(remotesock);
…end main loop…

close(mysock);
```

sockets - client example

```
mysock = socket(AF_INET,SOCK_STREAM,0);

connect(mysock,<remote address>);

write() or send() the request to the server
read() or recv() the response from the server

close(mysock);
```

inetd socket applications

- MPE INETD invokes servers with redirection:
  - fd 0 (stdin) redirected to the accept()ed socket
  - fd 1 (stdout) redirected to JINETD $STDLIST
  - fd 2 (stderr) redirected to JINETD $STDLIST
- dup2(0,1) for a more typical inetd environment
- just do your normal terminal I/O to stdin and stdout which are really network sockets

signals

- signal() & raise() are ANSI C, not POSIX.1
  - Use sigaction() instead
- Signal is generated, pending, delivered
  - Signal not delivered if process is executing in system code; signal is delivered upon exit of system code
- Process can:
  - Ignore the signal
  - Execute a signal-handling function; process resumes where it was interrupted
  - Restore the default action of the signal

signals - kill

- `int kill (pid_t` *`pid`*`, int` *`sig`*`);`
- sends a signal to another process
- `kill` shell command which calls the kill() function

error handling

- errno is a system global defined in <errno.h>
- Functions:
  - `char *strerror(int errnum);`
  - `void perror(const char *msg);`

```
if ( (fd = open(pathname, O_RDWR)) < 0)
    {
        /* errno already set by open() */
        perror("functionX(): open()");
        return -1;
     }
```

miscellaneous - system()

- `int system(const char *command);`
- passes *command* to the shell for execution
- all shell metacharacters will be acted upon, so use EXTREME caution when passing user-supplied data to system()! Note that popen() has the same issue.
    - `` `hacker command string` ``
    - `| hacker command string`
    - `> /some/file/to/destroy`

mpe intrinsics vs. posix functions

MPE intrinsic	POSIX function
ACTIVATE	exec
CATREAD	strerror
CLOCK	time
CREATEPROCESS	fork
FATHER	getppid
FCLOSE	close
FFILEINFO	fstat
FLOCK	fcntl
FOPEN	open
FPOINT	lseek
FREAD	read
FUNLOCK	fcntl
FWRITE	write

hp e3000

programming and posix

mpe intrinsics vs. posix functions (cont.)

MPE intrinsic	POSIX function
HPACDPUT ---------------->	chmod, fchmod
HPCICOMMAND -->	system
HPCIGETVAR ------------>	getenv
HPCIPUTVAR ------------->	putenv
HPERRMSG --------------->	strerror
HPFOPEN ------------------->	open
HPPIPE -------------------------->	pipe
KILL ---------------------------------->	kill
PAUSE -------------------------->	sleep
PRINT ---------------------------->	printf
PROCINFO ---------------->	getpid
PROCTIME ----------------->	times
QUIT ------------------------------>	exit, abort
WHO ---------------------------->	getlogin

additional programming topics

- Debugging Your Application
- Shell Scripts
- Regular Expressions
- Awk
- Security Pitfalls
- Development Tools
- GNU Tools
- Porting Wrappers

debugging your application - 1

- Add printf() statements in your code
  - use #ifdef DEBUG compile directive
- Add perror() statements in your code
  - use #ifdef DEBUG compile directive

```
    if ( (fd = open(pathname, O_RDWR)) < 0)
   {
      /* errno already set by open() */
    #ifdef DEBUG
       sprintf(msg, "functionX(): open(%s,
O_RDWR)”, pathname);
       perror(msg);
    #endif
      return -1;
    }
```

debugging your application - 2

- MPE System Debugger

```
shell/iX> callci “run ./program ;debug”
```

- Symbolic debugger - xdb (does not support gcc)
  - use -g switch during compile

    ```
    shell/iX> c89 -g ...
    ```

  - link with /SYS/LIB/XDBEND
    - first, as MANAGER.SYS:

      ```
      shell/iX> cd /SYS/LIB; ln -s XDBEND end.o
      ```

    ```
    shell/iX> c89 -o … /SYS/LIB/end.o
    shell/iX> xdb -h program
    ```

diff and patch commands

- diff - compares two files and reports on differences
  - -r option recursively compares two directory trees
  - -u option on GNU diff for making open-source patches

- patch - modifies files using diff output
  - can modify entire directory trees
  - saves rejected diff code in *.rej files
  - use GNU patch to handle diff -u format

shell programming

- Automate steps with a shell script hwcgi.sh

```
#!/bin/sh
gcc -c helloworld.c
ar -rv libhw.a helloworld.o
gcc -c hwcgimain.c
gcc -o hwcgi hwcgimain.o -L. -lhw
```

- Execute permission required to execute

```
shell/iX> chmod u+x hwcgi.sh
shell/iX> hwcgi.sh
```

- Special scripts: /etc/profile and .profile

shell interpreters

- the first line of a shell script specifies the interpreter to be run and the parameters if any, I.e.:

```
#!/bin/sh
```

- when a shell script is invoked by the shell or via exec(), the interpreter program is run with stdin redirected to the script file

posix shell command syntax

- `(cmd)` - execute cmd in a subshell
- `cmd1 | cmd2` - pipe cmd1 stdout to cmd2 stdin
- `cmd1 && cmd2` - execute cmd2 only if cmd1 returns zero exit status (true)
- `cmd1 || cmd2` - execute cmd2 only if cmd1 returns non-zero exit status (false)
- `cmd1; cmd2` - execute cmd1, then cmd2
- `cmd &` - execute cmd asynchronously

posix shell flow of control

-
  ```
  case word in
    pattern1) command1 ;;
    pattern2) command2 ;;
  esac
  ```

  execute the *command* of the first *pattern* matching *word*

-
  ```
  for variable in word1 word2 …; do
      command
  done
  ```

  for each *word*, set the *variable* to the *word* and execute the *command(s)*

posix shell flow of control (cont.)

-
  ```
  if command1; then
      command2
  elif command3; then
      command4
  else
      command5
  fi
  ```

traditional if-then-else; the elif and else clauses are optional

posix shell flow of control (cont.)

-
  ```
  until command1; do
     command2
  done
  ```

-
  ```
  while command1; do
     command2
  done
  ```

- the `break` and `continue` commands can be used to alter loop behavior

posix shell functions

- `function name {`
  ```
    command
  }
  ```
  or
  ```
  name() {
    command
  }
  ```
- treated just like any other command or script
- has a separate list of positional parameters
- may declare local variables

posix shell variables and arrays

- *`variable`*`=`*`value`* to assign a scalar variable
- *`variable`*`=`*`value command`* to make a variable assignment visible only to the new command env
- *`variable`*`[`*`index`*`]=`*`value`* to assign an array item
- `$`*`variable`* or `${`*`variable`*`}` to dereference a variable
- `$`*`variable`*`[`*`index`*`]` to deference an array item
- `${`*`variable`*`:-`*`default`*`}` for the non-null value of variable else default
- plus about 15 additional variations…
- `$?` - last exit status value
- `$$` - POSIX PID of the current shell
- `$!` - POSIX PID of the last asynchronous command

posix shell parameters

- `$0` - name of the shell script
- `$`*n* where n=1..9 - the nth positional parameter
- `$#` - the number of positional parameters
- `$@` - all positional parameters; if quoted, all positional parameters as separate quoted arguments
- `$*` - all positional parameters; if quoted, all positional parameters as a single quoted argument
- `shift` *n* - shift all positional parameters left by n positions

posix shell command substitution

- `` `command` `` (backquotes) or `$(command)`
- replaces the command substitution expression with the stdout output from the execution of *command*

-
  ```
  TIMESTAMP=`/bin/date`
  echo "$TIMESTAMP log event" >>logfile
  ```

posix shell file/directory substitution

- `~`*`user`* replaced by *`user's`* home directory
  - `cd ~MGR.APACHE/htdocs` = /APACHE/PUB/htdocs
- `* ? []` - pathname wildcards replaced by possibly multiple files/dirs
  - `*` - zero or more characters
  - `?` - one character
  - `[]` - group or range (first-last) of characters
  - `/*/PUB/foo.bar` - foo.bar in every PUB group on the machine
  - `/SYS/PUB/LOG????` - all system log files
  - `foo/[a-z]*` - all initially lowercase files in foo dir

posix shell i/o redirection

- <*file* - read stdin from *file*
- >*file* - write stdout to *file*
- >>*file* - append stdout to *file*
- 2>*file* - write stderr (2) to *file*
- 2>&1 - write stderr (2) to the same place as stdout (1)
- <<*name* - read stdin from the following lines of shell input until a line consisting of *name* is found

```
/bin/cat <<ALLDONE >file
here is some data
to be copied to a file
ALLDONE
```

posix shell escaping and quoting

- `\` - disregard the special meaning of the next character
- `'string'` - disregard the special meaning of all characters in the string
- `"string"` - disregard all special meanings except for command substitution and variable dereferencing
- bad: `callci run foo;info="bar"`
- good: `callci run foo\;info=\"bar\"`
- good: `callci 'run foo;info="bar"'`

posix shell callci command

- `callci` *`command_string`*
- used to invoke CI commands from the shell
- command_string gets passed to HPCICOMMAND
- callci uses CI I/O redirection in certain situations including batch jobs, so MPE commands that don't work with CIOR will fail
  - fails: `callci setvar variable value`
  - workaround: `callci mysetvar variable value`

posix shell test command

- `test` *`expression`* or `[` *`expression`* `]`
- exit status indicates the result of the expression:
  - 0 - true
  - 1 - false
  - 2 - expression syntax error
- `-f` *`file`* - true if the file exists
- `-d` *`file`* - true if the file is a directory
- *`string1`* `=` *`string2`* - true if strings are identical
- *`number1`* `-eq` *`number2`* - true if numbers are equal
- *`expr1`* `-a` *`expr2`* - AND relationship
- *`expr1`* `-o` *`expr2`* - OR relationship
- and many more...

regular expressions (regexp)

- the language of pattern matching
- man regexp for full syntax
- `.` - match any one character
- `^` - match the beginning of a line
- `$` - match the end of a line
- `[a-z]` - range match for lowercase
- `*` - match zero or more
- `+` - match one or more
- `?` - match one or zero
- `\(` and `\)` - grouping

awk programming - /bin/awk

- powerful regexp-based pattern matching and string manipulation
- great for file parsing and reformatting
- specify search patterns and associated actions
- full if-then-else logic and more
- better performance in certain applications compared to the POSIX shell because no forking will be done

potential posix security pitfalls

- loose or missing umask resulting in world- or group-writeable security
- files and directories rely on ACDs to implement security, and many MPE utilities may still result in ACDs being deleted
- setuid/setgid executables
- shell metacharacters like > or | or ` being parsed by popen() and system()
- user-supplied file names containing multiple upward directory references to reach the root and then downward to any file on the machine, I.e. ../../../SYS/PUB/HPSWINFO

development tools

- Edit files from another system
  - Samba - http: //jazz.external.hp.com/src/samba
- Development Environments
  - Whisper Technology - http://www.whispertech.com/pstudio.htm

gnu tools

- Downloadable software from:

  http:://jazz.external.hp.com/src/gnu/gnuframe.html

- Tools include:
  - gcc - C compiler
  - gxx or g++ - C++ compiler
  - gdb - debugger (port in progress)
  - gmake - for building software
  - gzip, gunzip - file compression and decompression
  - cvs - Concurrent Version System for software control

porting wrappers

- Downloadable software from:

  http: / / jazz.external.hp.com / src / px_wrappers / index.html

- Additional Functions:
  - Error reporting: pmpeerror, strmpeerror
  - Mapped regions: mmap, mprotect, msync, munmap
  - Sockets enabled: fcntl, fstat, stat
- Additional Libraries & Header Files
- Additional Commands:
  - ld, nm, nohup
  - Command wrappers: ftp, ipcs, ipcrm, ping, xdb

error handling with mpe intrinsics

- _mpe_errno, _mpe_intrinsic are system globals defined in <errno.h>
  - Requires _MPEXL_SOURCE compile directive to use
- Porting Wrappers has functions pmpeerror() & strmpeerror() plus header file <mpeerrno.h>

```
#include <mpeerrno.h>
#pragma intrinsic       FCONTROL
FCONTROL(_MPE_FILENO(fildes), 2, &dummy);
if ( ( ccode_return = ccode() ) != CCE )
  {
     errno = EINVAL;
     mpe_errno = ccode_return;
     mpe_intrinsic = FCONTROL_INTRINSIC;
#if defined(DEBUG) || defined(PRINT_ERROR)
     pmpeerror("functionX(): FCONTROL(2)");
#endif
     return -1;
   }
```

additional resources

- MPE/iX manuals:

  http://www.docs.hp.com

  - HP C/iX Library Reference Manual - function man pages
  - MPE/iX Developer's Kit Reference Manual - function man pages
  - MPE/iX Shell and Utilities User's Guide - commands, shell, vi, make
  - New Features of MPE/iX: Using the Hierarchical File System - commands
- Programming with examples:
  - "Advanced Programming in the UNIX Environment" by W. Richard Stevens

    http://www.kohala.com/start/apue.html

    - directory util/apue in Porting Wrappers contains Stevens' main header file and library

additional resources (cont.)

- POSIX
  - “POSIX Programmer's Guide” by Donald Lewine http://www.oreilly.com/catalog/posix/
  - “The POSIX.1 Standard - A Programmer’s Guide” by Fred Zlotnick
  - POSIX Specifications from IEEE - very detailed http://standards.ieee.org/catalog/posix.html#gen22
- make
  - “Managing Projects with make” by Andrew Oram and Steve Talbott

    http://www.oreilly.com/catalog/make2/

additional resources (cont.)

- `:XEQ POSIXCBT.LSN.SYS` - a basic POSIX tutorial bundled in FOS since 5.0 (primarily covers HFS topics)

join the hp3000-l community!

- Available as a mailing list and as the Usenet newsgroup comp.sys.hp.mpe
- In-depth discussions of all things HP e3000
- Talk with other people using POSIX on MPE
  - seek advice, exchange tips & techniques
- Keep up with the latest HP e3000 news
- Interact with CSY
- http://jazz.external.hp.com/papers/hp3000-info.html