1 Function Pointers

`int (*fp)(int, char*)` - Declares fp to be a pointer to a function which takes int and char* arguments and returns int

`void (*fp2[10])(double)` - Declares fp2 to be an array (of size 10) of pointers to functions taking a double parameter and returning nothing

`int (*fp3)()` - Declares fp3 to be a pointer to a function returning int. Argument types unknown and won't be checked by the compiler (Up to programmer to use this correctly).

2 Storage classes

2.1 auto

- Variable has local (automatic) extent, i.e. removed at end of block
- Permitted within a block only (i.e. not top level)
- This is the default so rarely seen

2.2 extern

- Variable/function is external to all functions, i.e. can be accessed by name by any function
- Globally accessible - linker must know about the name
- Must be defined once somewhere (can be declared anywhere)

2.3 register

- Hint to compiler to put variable in a register, otherwise like auto

2.4 static

- Name is only accessible in this file (i.e. not exported to linker)
- For variables - extent is static - variable lasts for life of program

3 5 Views of OS

1. Hardware
2. Operating System Designer
3. Application Programmer
4. End-User
5. System Administrator

4 Environment Variables

HOME - full pathname of home directory
PATH - colon separated list of pathnames to search for commands
USER - your username
SHELL - full pathname of your login shell

4.1 Local Variable

`courseCode=CSSE2310`

4.2 Environment Variable

`export courseCode`

5 Metacharacters

5.1 WildCards

***** - zero or more characters
? - any single character

5.2 Comment

- start of comment (goes till end of line)

5.3 Running Commands

& - run command in background
; - used to separate commands
‘command‘ - substitute result of running command

5.4 Variable Substitution

$varname - substitute value of variable

5.5 SubShell

(...commands...) - execute commands in a sub-shell

5.6 Conditional sequences

|| - execute command if previous command failed
&& - execute command if previous one succeeded
A success is when the program returns an exit code 0

5.7 Redirection and Pipes

| - pipe, output of one program sent to the input of the next
> - send stdout to file
< - write stdin from file
» - append stdout to file

5.8 Quoting

Single Quotes - inhibit wildcard replacement, variable substitution, command substitution.
Double Quotes - inhibit wildcard replacement only.
Can also use backslashes.

6 Shell Scripting

6.1 Shell Arithmetic

Use `expr EXPRESSION` command.
Result is printed to stdout.

6.2 Shell Conditional

Use `test EXPRESSION` or `[ EXPRESSION ]`. `EXPRESSION` follows `INT1 option INT2` where option can be: `-eq` (equals), `-ge` (greater than or equal), `-gt` (greater than), `-le` (less than or equal), `-lt` (less than), `-ne` (not equal). Result is exit code 1 for false and 0 for true.

6.3 Control Structures

For Loop

```
for name [ in word...  ]
do
commands...
done
```

If words omitted, uses script arguments.

While Loop

```
while command1
do
command2
done
```

If Then Else

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

Other Control Structures

Case. Until - do - done. Done.

7 Built-in Shell Variables

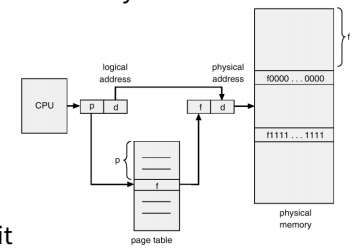
$$ - process ID of shell
$0 - name of the shell script
$1...$9 - command line arguments
$* - all the command line arguments
$# - number of command line arguments (excludes command name)
$? - exit status of last command
$! - process ID of last background command

8 Paging

One of several methods of implementing virtual memory. Divide physical memory into fixed-sized blocks called page frames (size is power of 2, usually between 512 bytes and 8192 bytes). Divide logical memory into blocks of same size called pages. Keep track of all free frames. To run a program of n pages, need to find n free frames and load program. Set up a page table to translate logical to physical addresses.

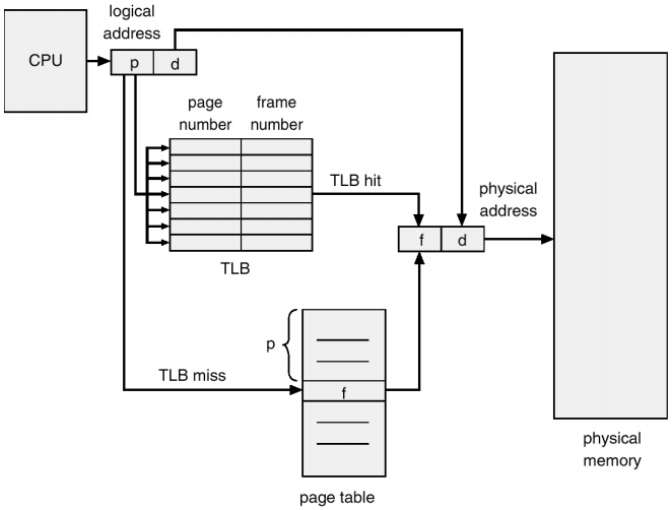
8.1 Address Translation Scheme

Page Number (p) - used as an index into a page table which contains base address of each page in physical memory
Page Offset (d) - combined with base address to define the physical memory address that is sent to the memory unit



8.2 Implementation of Page Table

Page table is kept in main memory. The two memory access problem can be solved by the use of a special fast-lookup hardware cache called associative memory or translation look-aside buffers (TLBs)



Page Faults

If the TLB entry doesn't exist, then load it into the table

8.3 Memory Protection

Memory protection implemented by associating protection bit with each frame.

8.4 Segmentation Faults

Segmentation Fault arises when: Accessing invalid memory page, Trying to write to a read-only page.

8.5 Shared Pages

Shared code

One copy of read-only code shared among processes. Shared code must appear in same location in the virtual address space of all processes.

Private code and data

Each process keeps a separate copy of the code and data. The pages for the private code and data can appear anywhere in the virtual address space.

8.6 User-space vs Kernel

The operating system controls the address ranges (pages) a process can use. It does not decide how that space is used. Management of those pages is the responsibility of the process (Usually vie standard libraries).

9 Memory-Related Bugs

- Dereferencing bad pointers
- Reading uninitialized memory
- Overwriting memory
- Referencing nonexistent variables
- Freeing blocks multiple times
- Referencing freed blocks
- Failing to free blocks

10 Processes

10.1 fork

`int fork(void)` - creates a new process. Returns 0 to the child process else return child's pid.
Common for the parent to be the only one to continue forking

10.2 exit

`void exit(int status)` - exits a process (status 0 for no error)
`_exit()` - will not run the registered exit functions

10.3 atexit

`atexit(void (*func)(void))` - registers functions to be executed upon exit. Functions called in reverse order of registration.

10.4 wait

`int wait(int child_status)` - suspends process until child process terminates. Return the exit code, if child_status is `NULL` then any child process.

10.5 waitpid

`waitpid(pid, &status, options)` - can wait for a specific process. Various options (default is 0)

10.6 exec

`int execl(char *path, char *arg0, char *arg1, ..., 0)` - loads and runs executable at path with args arg0, arg1... (path is the complete path of the executable, arg0 becomes the name of the process).

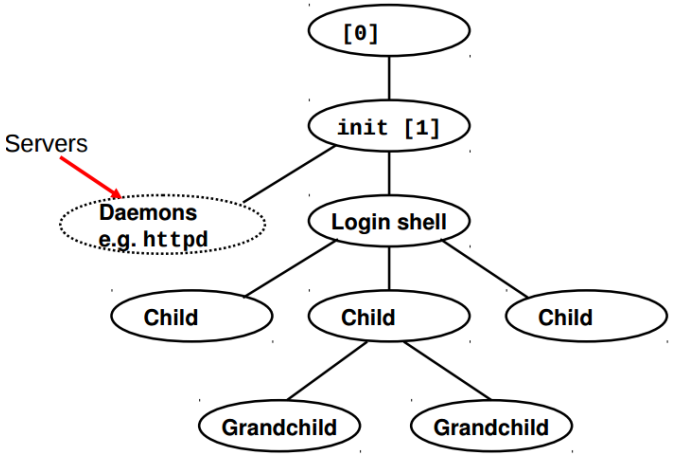
exec variations

`execl()`, `execve()`, `execv()`, `execlp()`, `execle()`, `execvp()`
l - arguments directly in call (list)
v - arguments in array (vector)
p - use PATH to find program
e - provide environment definition

10.7 Resource Sharing

- Parent and children share all resources
- Children share subset of parent's resources
- Parent and child share no resources

10.8 Unix Process Hierarchy



11 Files

11.1 Unix File Types

Regular file

Binary or text file. Unix does not know the difference

Directory file

A file that contains the names and locations of other files

Links

Symbolic links to other files

Character special and block special files

Terminals (character special) and disks (block special)

FIFO (named pipe)

A file type used for interprocess communication

Socket

A file type used for network communication between processes

11.2 Sharing Files

A child process inherits its parent's open files. `refcnt` counts the number of references.

12 Piping

12.1 I/O Redirection

dup2

`dup2(oldfd, newfd)` - copies per-process descriptor table entry oldfd to entry newfd.

12.2 Example

```
int fdInput[2], fdOutput[2];
if (pipe(fdInput) == 0 && pipe(fdOutput) == 0) {
    int pid = fork();
    if (pid == 0) {
        //Child
        close(fdOutput[1]);
        close(fdInput[0]);
        dup2(fdOutput[0], STDIN_FILENO);
        dup2(fdInput[1], STDOUT_FILENO);
        if (execl(processName, processName, arg1,
                arg2, NULL) == -1) {
            error(SUBPROCESS_ERROR);
        }
    } else {
        //Parent
        childPid = pid;
        close(fdInput[1]);
        close(fdOutput[0]);
        input = fdopen(fdInput[0], "r");
        output = fdopen(fdOutput[1], "w");
    }
}
return
```