CSC 360: Operating Systems

Tutorial #2

Image: https://commons.wikimedia.org/wiki/File:Seashells_North_Wales_1985.jpg

UNIX Shells and Process Creation

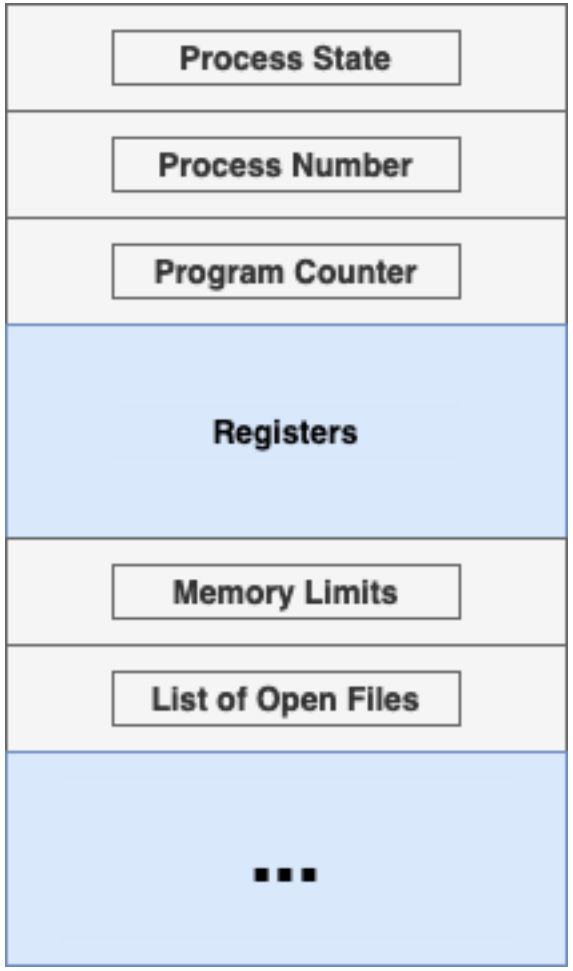
What is a Shell?

- Before there were graphical user interfaces, users had to write instructions to the computer on a keyboard.
- The earliest form of this was a teletype which was a typewriter with input and output. (We still have ASCII values 0x07 for bell and 0x0E for carriage return).
- A shell is a *command interpreter* for the operating system.
- It allows the user to execute programs as well as built in commands.
- When a user types a command, the shell parses it. It then looks in the $PATH directories and built-in commands to determine what to do.
- When we run a program from the shell it spawns a new process and waits for it to finish.
- The shell is a user-level program that requests the OS to perform tasks on its behalf
- Examples: BASIC interpreters, Windows CMD.exe, Linux: sh, cshell, bash etc.

What is a Process?

- A process is an instance of a program that is currently executing on the system
- A program is simply a list of instructions that is stored permanently (on disk)
- When a process starts, these instructions are loaded into main memory (RAM). The operating system allocates each process its own address space.
- This ensures that processes cannot access the memory used by other processes
- Since many processes may be running at the same time, the OS must manage the sharing of system resources between processes
- The scheduler maintains state queues which determine whether a process is currently running on the CPU, waiting for the CPU, or doing something else. (just linked-lists etc.)
- Information about each process is stored in a *Process Control Block* (just a big struct)

What is in the PCB?



- The Linux kernel uses task_struct to implement the PCB
- It is simply a struct with various settings
- Some of these are:

```
pid_t pid;                    // Used to identify the process
unsigned int cpu;             // The current CPU
int exit_code;                // Return value of process
struct task_struct __rcu *parent      //Parent PCB
```

https://github.com/torvalds/linux/blob/master/include/linux/sched.h

How are processes created?

- Processes are started by *cloning* themselves from other processes
- A parent process will call the fork() system call.
- This causes a trap into kernel mode. The OS then runs appropriate code from a lookup table of system calls.
- The OS will then make an exact copy of the parent process's address space and PCB then adds the child process to the ready queue (starts execution).
- The only difference is that the child process has a different PID from its parent
- Both processes will resume executing from the instruction immediately following fork()
- In the parent, fork() returns the child's PID
- In the child, fork() returns 0
- This is how we differentiate between them

```c
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdio.h>

int main()
{
  pid_t pid;

  pid = getpid();

  printf("The PID of this process is: %d\n", pid);

  pid = fork();
  if (pid == 0) { //Returns PID of 0
    // Child process
    printf("This is the child process with fork returning PID: %d\n", pid);
    sleep(10); //Just busy waits for 10 seconds
  } else if (pid < 0) {
    // Something went wrong
    printf("Error forking:");
  } else { //Returns PID of child
    // Parent process
    printf("This is the parent with fork returning PID: %d\n", pid);
    wait(NULL); //Wait for child process to terminate before exiting
  }
}
```

```
The PID of this process is: 12511
This is the parent with fork returning PID: 12513
This is the child process with fork returning PID: 0
```

How do we run a new program?

- We must use the exec() system call.
- After we have created a *cloned* process using fork()
- This copies the instructions (code) of a new program into the address space of the process that it is called from.

```c
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>

int main()
{
    pid_t pid;

    /* fork a child process */
    pid = fork();

    if (pid < 0) { /* Something went wrong */
      fprintf(stderr, "Fork failed\n");
      return 1;
    }
    else if (pid == 0) { /* Child process */
      execlp("/bin/ls", "ls", NULL);
    }
    else { /* Parent process */
      /* Parent will wait for the child to complete */
      wait(NULL);
      printf("Child complete\n");
    }

    return 0;
}
```

What Should the Shell Do?

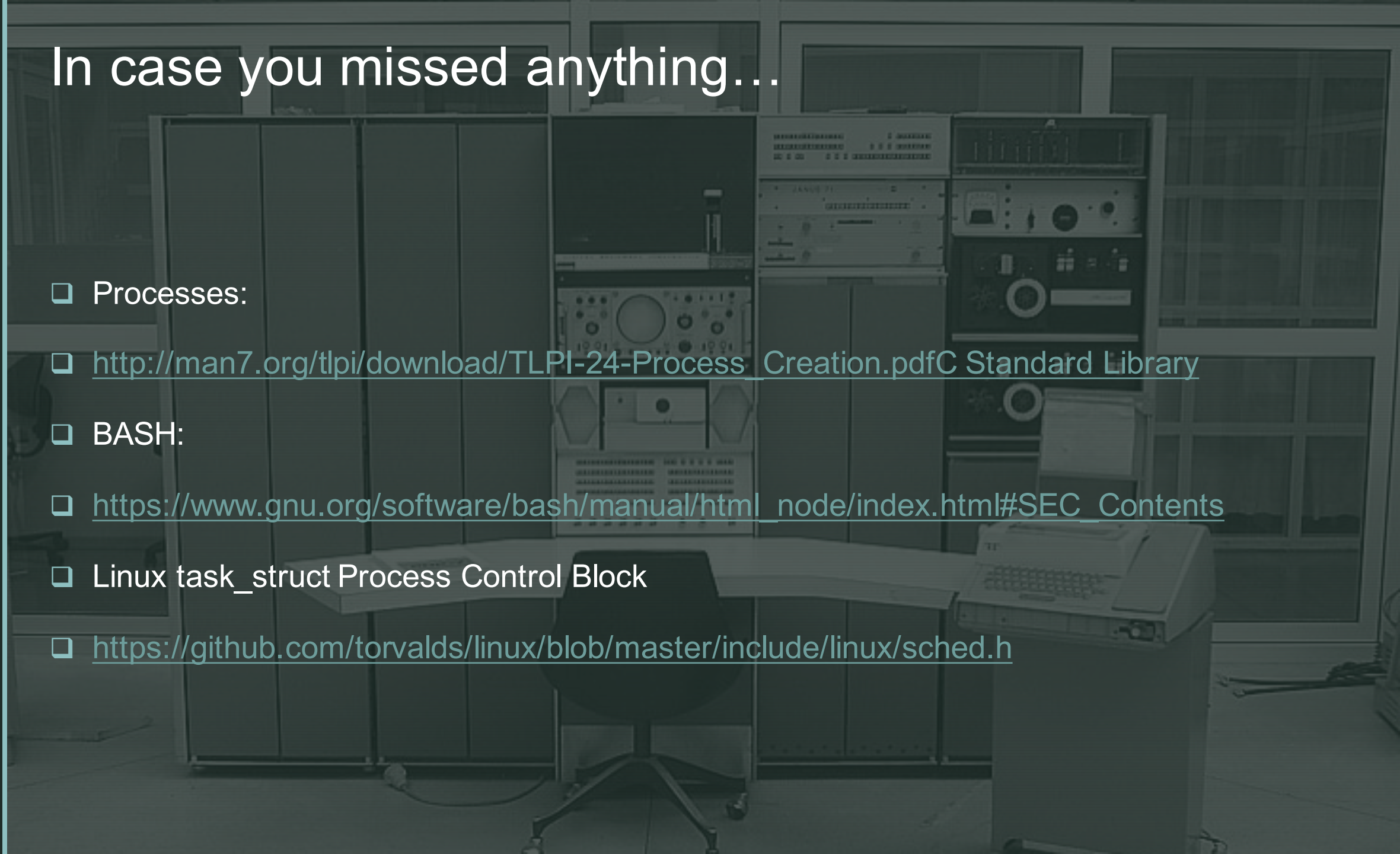
- Read an input string from the user
- Split the string into *tokens* and store in an array
- Analyze the token array to form commands, options and arguments
- Execute the command (built-in or look in path for program to run)
- Repeat…

Quiz

- Draw a tree diagram of the output of the following code:

```
<secret!>
```

In case you missed anything…



- Processes:
- http://man7.org/tlpi/download/TLPI-24-Process_Creation.pdfC Standard Library
- BASH:
- https://www.gnu.org/software/bash/manual/html_node/index.html#SEC_Contents
- Linux task_struct Process Control Block
- https://github.com/torvalds/linux/blob/master/include/linux/sched.h