SE350: Operating Systems

Tutorial: The Programming Interface
Main Points

- Creating and managing processes
  - fork, exec, wait
- Example: implementing a shell
Shell

• A shell is a job control system
  • Allows programmer to create and manage a set of programs to do some task
  • Windows, MacOS, Linux all have shells

• Example: to compile a C program
  
  cc –c sourcefile1.c
  cc –c sourcefile2.c
  ln –o program sourcefile1.o sourcefile2.o
Windows CreateProcess

• System call to create new process to run a program
  • Create and initialize process control block (PCB) in kernel
  • Create and initialize a new address space
  • Load the program into the address space
  • Copy arguments into memory in the address space
  • Initialize hardware context to start execution at “start”
  • Inform the scheduler that the new process is ready to run
Windows CreateProcess API (simplified)

if (!CreateProcess(
    NULL,  // No module name (use command line)
    argv[1],  // Command line
    NULL,  // Process handle not inheritable
    NULL,  // Thread handle not inheritable
    FALSE,  // Set handle inheritance to FALSE
    0,  // No creation flags
    NULL,  // Use parent's environment block
    NULL,  // Use parent's starting directory
    &si,  // Pointer to STARTUPINFO structure
    &pi ))  // Pointer to PROCESS_INFORMATION structure

UNIX Process Management

- UNIX fork
  - System call (without any arguments) to create a copy of the current process, and start it running

- UNIX exec
  - System call to change the program being run by the current process

- UNIX wait
  - System call to wait for a process to finish

- UNIX signal
  - System call to send a notification to another process (e.g. SIGKILL, SIGINT)
UNIX Process Management

```c
pid = fork();
if (pid == 0)
  exec(...);
else
  wait(pid);

main () {
  ...
}
```
Question: What Does This Code Print?

1. int child_pid = fork();

2. if (child_pid == 0) {
   // I'm the child process
3.      printf("I am process #\%d\n", getpid());
4.      return 0;
5. } else {
   // I'm the parent process
6.      printf("I am parent of process #\%d\n", child_pid);
7.      return 0;
8. }

Possible output:
I am parent of process 495
I am process 495

Another less likely but still possible output:
I am process 456
I am parent of process 456
Questions

• Can UNIX fork() return an error? Why?

• Can UNIX exec() return an error? Why?

• Can UNIX wait() ever return immediately? Why?
Implementing UNIX fork

Steps to implement UNIX fork

• Create and initialize the process control block in kernel
• Create a new address space
• Initialize the address space with a copy of the entire contents of the address space of the parent
• Inherit the execution context of the parent
  • E.g., any open files
• Inform the scheduler that the new process is ready to run
Implementing UNIX exec

- Steps to implement UNIX exec
  - Load the program into the current address space
  - Copy arguments into memory in the address space
  - Initialize the hardware context to start execution at “start”
Implementing a Shell

1. char *prog, **args;
2. int child_pid;

3. // Read and parse the input a line at a time
4. while (readAndParseCmdLine(&prog, &args)) {
5.    child_pid = fork(); // create a child process

6.    if (child_pid == 0) {
7.        exec(prog, args); // I'm the child process. Run program
8.    } else {
9.      } else {
10.         wait(child_pid); // I'm the parent, wait for child
11.         return 0;
12.    }
13. }

Acknowledgment

• This lecture is a slightly modified version of the one prepared by Tom Anderson