Javier Garcia

Operating Systems Midterm Exam

1. For each of the following process state transition, say whether the transition is legal and how the transition occurs or why it cannot.
   1. Change from thread state WAIT to thread state RUNNING
      1. This transition is illegal because a process in the WAIT/BLOCKED state must go through the READY state before it can run. It cannot be schedules from a blocked state to a running state directly.
   2. Change from thread state RUNNING to thread state WAIT
      1. This is a legal transition. When a RUNNING process initiates I/O or any other blocking operation, it then becomes a state of BLOCKED/WAIT.
   3. Change from thread state READY to thread state WAIT
      1. This is an illegal transition, because a process must go from READY to RUNNING, but it cannot go back to WAIT/BLOCKED until it is scheduled and executed.
2. Write a program that opens a file (with the open () system call) and then calls fork () to create a new process. Can both the child and parent access the file descriptor returned by open ()? What happens when they are writing to the file concurrently, i.e., at the same time?
   1. Int main() {

int fileDescriptor = open(“filename.txt”)

int rc = fork();

if (rc == 0) {

write(fileDescriptor, “This is the child process writing. \n”, 36);

}

else {

write(fileDescriptor, “This is the parent process writing. \n”, 37);

}

close(fileDescriptor);

}

Yes, both the parent and the child can access the file at the same time, since the call fork() duplicates the process, and all the variables associated with it. They then execute at the same time and write to the file at the same time, although one would appear to be finished before the other.

1. Write another program using a fork (). The child process should print “hello”; the parent process should print “goodbye”. You should try to ensure that the child process always prints first; can you do this without calling wait () in the parent?
   1. int main() {

pid\_t pid = fork();

if(pid == 0) {

printf(“Hello\n”);

}

else {

waitpid(pid, NULL,0);

printf(“Goodbye\n”);

}

return 0;

}

1. Write a program that creates a child process, and then in the child closes standard output (STDOUT FILENO). What happens if the child calls printf () to print some output after closing the descriptor?

int main() {

int process = fork();

if (process == 0) {

close(STDOUT\_FILENO);

printf("This text will not print.\n");

} else {

printf("This text will print.\n");

}

return 0;

}

Here, the parent process will print to the console, however, because the child process closed the standard output, it will not be able to print to the console.

1. Consider the following piece of C code:

void main ( ) {

fork ( );

fork ( );

exit ( );

}

How many child processes are created upon execution of this program?

There will be four new processes created from the two forks. Two of them will be parents and the other two would be the children, each pair containing one of each.