Operating Systems Lab – AIM2231 (AY : 2024-25)

Upendra Singh
Assistant Professor
Department of AIML, School of Computer Science and Engineering
Manipal University Jaipur.

Week 5 : System Calls

Demonstrate the use of file system calls.

Outline

- Process Creation
- Process Termination
- How to use process related System Calls
- Examples
- Exercise

Process Creation

- **Parent** process creates **children** processes, which, in turn, create other processes, forming a **tree** of processes
- Generally, a process identified and managed via a **process identifier** (**pid**)

Process Creation (Cont.)

When a process creates a new process, two possibilities for execution exist:

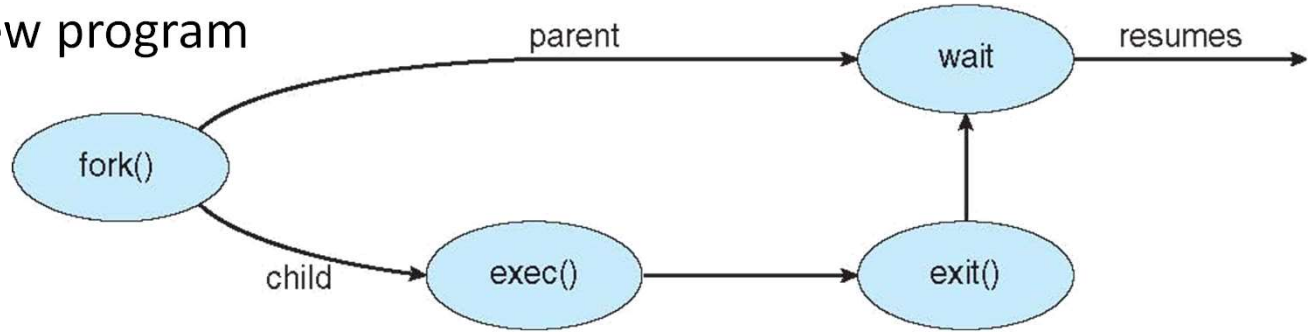
- The parent continues to execute concurrently with its children.
- The parent waits until some or all of its children have terminated.

There are also two address-space possibilities for the new process:

- The child process is a duplicate of the parent process (it has the same program and data as the parent).
- The child process has a new program loaded into it.

Process Creation (Cont.)

- Address space
  - Child duplicate of parent
  - Child has a program loaded into it
- UNIX examples
  - **`fork()`** system call creates a new process
  - **`exec()`** system call used after a **`fork()`** to replace the process' memory space with a new program



Process Termination

- Process executes the last statement and then asks the operating system to delete it using the **`exit()`** system call.
  - Returns status data from child to parent (via **`wait()`**)
  - Process resources are deallocated by the operating system
- The Parent may terminate the execution of the children's processes using the **`abort()`** system call. Some reasons for doing so:
  - Child has exceeded allocated resources
  - Task assigned to a child is no longer required
  - The parent is exiting and the operating system does not allow a child to continue if its parent terminates

Process Termination

- Some operating systems do not allow a child to exist if its parent has terminated. If a process terminates, then all its children must also be terminated.
  - **cascading termination.** All children, grandchildren, etc. are terminated.
  - The termination is initiated by the operating system.
- The parent process may wait for the termination of a child process by using the **`wait()`** system call. The call returns status information and the PID of the terminated process

```
pid = wait(&status);
```

Using fork() and wait()

1. The **`fork()`** system call is used to create a new child process.
2. The return value of **`fork()`** determines whether the process is the parent or the child:
   A. If **pid == 0**, it's the child process.
   B. If **pid > 0**, it's the parent process.
   C. If **pid < 0**, fork failed.
3. The child process prints its PID and its parent's PID, then terminates using **`exit(0)`**.
4. The parent process calls **`wait(NULL)`**, which makes it wait until the child process terminates.
5. Once the child exits, the parent prints a message and completes execution.

Using exec()

1. `fork()` creates a child process.
2. The child process prints a message and then calls `execl()`, which replaces the child's process with the /bin/ls program.
3. The `execl("/bin/ls", "ls", "-l", NULL);` command runs ls -l (list directory contents).
4. If exec succeeds, the rest of the child process code does not execute.
5. If exec fails, `perror("exec failed")` prints an error.
6. The parent waits `(wait(NULL))` for the child to finish and then prints a message.

Using dup() for File Descriptor Duplication

1. The `open()` function opens (or creates) `output.txt` in write-only mode.
2. If the file cannot be opened, the program prints an error and exits.
3. `dup2(file, STDOUT_FILENO);` redirects stdout to the file.
4. The file descriptor is closed with `close(file)`, but `stdout` remains redirected.
5. `printf()` writes to the file instead of the terminal.
6. When you check output.txt, you will find the printed text inside.

Using exit()

1. **fork()** creates a child process.
2. If it's the child, it prints a message and calls exit(42), terminating with exit status 42.
3. The parent waits for the child to exit using **wait(&status).**
4. **WEXITSTATUS(status)** extracts the exit status (42) and prints it.

Using pause()

`signal(SIGINT, signal_handler);` sets up a handler for `Ctrl+C (SIGINT).`

The program prints its PID and calls `pause()`, which makes it wait indefinitely.

When the user presses `Ctrl+C`, the `signal_handler()` function runs.

The handler prints the received signal and exits the program.

Summary of System Calls Used

System Call	Description
fork()	Creates a new child process.
exec()	Replaces the process with a new program.
dup2()	Redirects file descriptors (useful for output redirection).
exit()	Terminates a process with a specific status.
wait()	Makes a parent process wait for a child to terminate.
pause()	Suspends execution until a signal is received.

Examples Programs

Source Code Folder Link

https://mujcampus-my.sharepoint.com/:f:/g/personal/upendra_singh_jaipur_manipal_edu/EjL1pWMmp-tNjcaKHMTjfdYBJ_sIRyew-7CBaRQ08D5k6g?e=HLB5xg

Exercises 1: Using exit() with Parent and Child Coordination

Problem Statement: Write a program that:

1. Creates two child processes.
2. Each child process generates a random integer and exits with that number as the exit status.
3. The parent process waits for both children and prints their exit statuses.
4. The parent process exits with the sum of both children's exit statuses.

Hints:

1. Use rand() % 100; to generate random numbers.
2. Extract exit statuses using WEXITSTATUS(status).
3. Parent should return exit(sum_of_child_statuses).

Exercises 2: File Redirection and Process Communication

Problem Statement: Write a program that:

1. Creates a child process.
2. The child process redirects its stdout to a file named "child_output.txt".
3. The child process then runs ls -l using execlp(), so the output is written to the file instead of the terminal.
4. The parent process waits for the child and then reads the contents of "child_output.txt" and displays them.

Hints:

1. Use dup2() to redirect stdout in the child.
2. Use fopen() or read() in the parent to display the file contents.


Thank
you