

CS 3305A

Process

Lecture 2

Sept 13th 2023

Copyright Notice:

1. Lecture slides are being provided with permission from the copyright for in-class (CS3305A) use only. The slides must not be reproduced or provided to anyone outside of the class.
2. All download copies of the slides and/or lecture recordings are for personal use only.
3. Students must destroy these copies within 30 days after receipt of final course evaluations

Topics

- ❑ Process Basics
- ❑ How to create a process using C
 - ❑ fork() system call
- ❑ Programming Demo: Parent & Child process
 - ❑ fork1.c
 - ❑ fork2.c

Process

- ❑ A **process** is an instance of an application running
- ❑ As a process executes it changes state.
- ❑ Possible states:
 1. **New**: The process is created
 2. **Running**: Instructions are being executed
 3. **Waiting**: The process is waiting for some event to occur e.g., I/O completion
 4. **Terminated**: Process has finished execution

Process Control Block

- ❑ Each process is represented in the operating system by a **process control block (PCB)**

- ❑ PCB includes information such as:

- ❑ Process Identifier (PID)
- ❑ Process state
- ❑ Program counter
- ❑ CPU registers
- ❑ CPU-Scheduling information
- ❑ Memory-Management information
- ❑ I/O status information

PCB

- ❑ Process Identifier (PID)
- ❑ Process state
- ❑ Program counter
- ❑ CPU registers
- ❑ CPU-Scheduling information
- ❑ Memory-Management info
- ❑ I/O status info

The Concept of Fork

- ❑ The Unix system call for process creation is `fork()`
- ❑ The `fork` system call creates a child process that is a duplicate of the parent.
 - ❑ Child inherits state from parent process
 - ❑ Same program instructions, variables have the same values, same position in the code.
 - ❑ Parent and child have separate copies of that state
 - ❑ They are stored in separate locations in memory
 - ❑ Important: updating the value of a variable inside the child **will NOT update** that variable in the parent, and vice-versa.

Fork System Call

- ❑ If `fork()` succeeds it returns integer value > 0 (which is the ID of the child) to the parent and returns 0 to the child
- ❑ If `fork()` fails, it returns -1 to the parent (no child is created)
- ❑ `pid_t` data type represents process identifiers
- ❑ Other calls:
 - ❑ `pid_t getpid()` - returns the ID of calling process. Call is always successful.
 - ❑ `pid_t getppid()` - returns the ID of parent process. Call is always successful.

fork() example

Parent's ID: 700

Child's ID: 701

```
int main ()
{
  pid_t i, j, pid;
  pid=fork();

  if (pid < 0) //unsuccessful
    printf("fork
    unsuccessful");

  if (pid > 0) //parent
  {
    i = getpid();
    printf("\n I am parent
    with PID %d", i)
  }

  If (pid==0) //child
  {
    i = getpid();
    j = getppid();
    printf("\n I am a
    child with PID %d and my
    parent's PID is %d", i, j);
  }
}
```

```
{
  pid_t i, j, pid;
  pid=fork();

  If (pid < 0) //unsuccessful
    printf("fork unsuccessful");

  if (pid > 0) //parent
  {
    i = getpid();
    printf("\n I am parent with
    PID %d", i)
  }

  if (pid==0) //child
  {
    i = getpid();
    j = getppid();
    printf("\n I am a
    child with PID %d and my
    parent's PID is %d", i, j);
  }
}
```

Parent creates its child

pid = 0

i = 701
j = 700

pid = 701

i = 700