

1. The operating system introduces processes to allocate the use of system resources, such as the CPU and memory, among multiple tasks. A process is a program in execution. It is an instance of a computer program that is being executed by one or many threads. It can be divided into 4 sections: stack, heap, data, and text. This allows for efficient utilization of resources, as well as providing a way to separate and isolate different tasks and applications from each other, thereby increasing stability and reliability.

In terms of process management, an operating system should provide the following functions to achieve cooperation and coordination among concurrent processes:

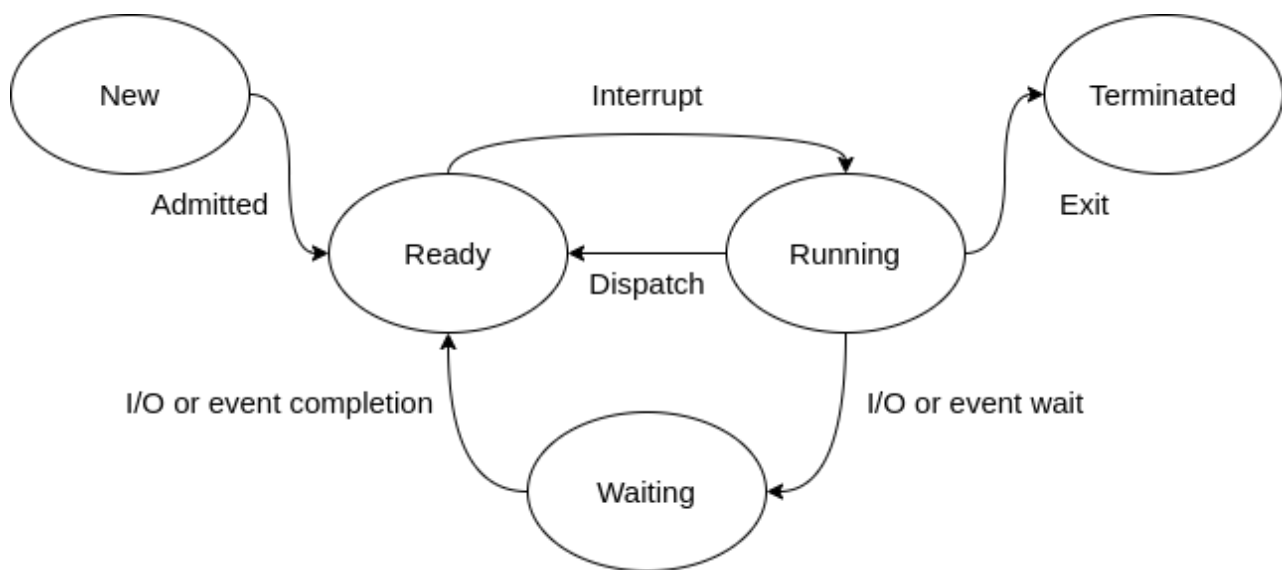
- Process creation and deletion: The operating system should allow for the creation and deletion of processes as needed.
- Process scheduling: The operating system should schedule the execution of processes, deciding which process should run at any given time, and allocating CPU time to each process in a fair and efficient manner.
- Inter-process communication: The operating system should provide mechanisms for processes to communicate and synchronize with each other, such as semaphores, message passing, and shared memory.
- Deadlock handling: The operating system should detect and resolve deadlocks, which occur when multiple processes are waiting for each other to release resources, causing a system-wide blockage.
- Memory management: The operating system should manage the allocation of memory to processes, ensuring that each process has enough memory to run, and that processes do not interfere with each other's memory.

2. A program is a set of instructions that are written in a language that can be executed by a computer. It is a static entity that exists in a stored form until it is executed. A program becomes a process when it is loaded into memory and begins execution.

The main differences between processes and programs are:

- State: A program is a passive entity with no state, while a process is an active entity that has a state and can be in different states such as running, blocked, or ready.
- Execution: A program is executed only when it is loaded into memory and becomes a process. A process can be executed concurrently with other processes and can be interrupted or rescheduled by the operating system.
- Memory: A program is stored in a file on a disk, while a process has its own memory space, including code, data, and stack.
- Interaction with the operating system: A program interacts with the operating system only when it is loaded into memory and becomes a process. A process can interact with the operating system for various services, such as accessing I/O devices or requesting memory.
- Resources: A program does not have resources, while a process has resources, such as CPU time, memory, and file descriptors, that are allocated by the operating system.

3.



Source: Zouaoui, S., Boussaid, L. and Mtibaa, A. (2019). Priority based round robin (PBRR) CPU scheduling algorithm. *International Journal of Electrical and Computer Engineering (IJECE)*, 9(1), p.190. doi:10.11591/ijece.v9i1.pp190-202.

4. The output at LINE A will be "Parent: value = 8".

The fork() function creates a new process by duplicating the calling process. In this case, the parent process will be the one that executes the line `printf("Parent: value = %d\n",value);`. The child process increments the value of value by 15, but since the child process terminates immediately after the increment, the value of value in the parent process remains unchanged and is still equal to 8.

The wait(NULL) function causes the parent process to wait for the child process to complete before continuing execution. Once the child process terminates, the parent process prints the value of value, which is 8.

5. A context switch is the process of storing and restoring the state of a process so that execution can be resumed from the same point later. The following are the main actions taken by the kernel during a context switch between processes:

Saving the current process state: The kernel saves the values of the CPU registers, including the program counter, stack pointer, and any other register that holds valuable information about the current process's state.

Selecting the next process: The kernel selects the next process to run based on its scheduling algorithm, which determines the order in which processes are executed.

Loading the next process state: The kernel loads the saved state of the next process into the CPU registers, including the program counter, stack pointer, and other important register values.

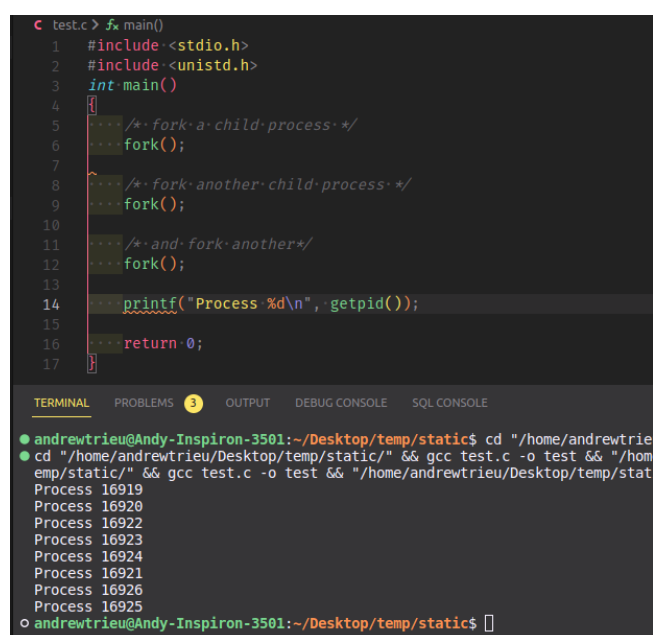
Switching the address space: The kernel switches the address space of the process to the address space of the next process. This is done to ensure that the process only accesses its own memory and cannot interfere with the memory of other processes.

Resuming the execution of the next process: The kernel resumes the execution of the next process by transferring control to the instruction pointed to by the program counter.

6. Eight processes are created by the following program, including the initial parent process.

Each time the `fork()` function is called, it creates a new child process that is an exact copy of the parent process. The child process starts executing from the line immediately following the `fork()` call. In this program, `fork()` is called three times, creating two child processes each time, for a total of $2^3 = 8$ processes. The initial parent process and seven child processes are created.

To confirm this, we could add a call to `printf()` at the end to print the process ID of each process and observing the output. The process ID of each process will be unique, indicating that multiple processes have been created. Example:



```
1  test.c > fx main()
2  #include <stdio.h>
3  #include <unistd.h>
4  int main()
5  {
6      /* fork a child process */
7      fork();
8      /* fork another child process */
9      fork();
10     /* and fork another */
11     fork();
12     printf("Process %d\n", getpid());
13     return 0;
14 }
```

```
andrewtrieu@Andy-Inspiron-3501:~/Desktop/temp/static$ cd "/home/andrewtrieu/Desktop/temp/static/" && gcc test.c -o test && "/home/andrewtrieu/Desktop/temp/static/" && gcc test.c -o test && "/home/andrewtrieu/Desktop/temp/static/" && ./test
Process 16919
Process 16920
Process 16922
Process 16923
Process 16924
Process 16921
Process 16926
Process 16925
andrewtrieu@Andy-Inspiron-3501:~/Desktop/temp/static$
```