CSc 360: Operating Systems (Spring 2022)

Written Assignment 1 (W1)

Alex Holland V00

1.

(a)

- User mode is not given access to the memory or hardware.
- If the User mode wants to access system resources, it must use a system API.
- The kernel mode is the privileged mode where the process is given unrestricted access to the memory or hardware.
- Crashes that occur in Kernel mode are much more serious then User mode.

(b)

- Kernel mode is reserved for the most trusted, low level operating system instructions.
- In User mode, the operating system can prevent many crashes since a crash should only affect the running user program.
- A User mode can also prevents a users program from 'accidentally' overwriting the operating system with user data.

(c)

- Mode switch refers to the process of switching the bit from 1 for user mode to 0 for kernel mode, and vice versa. This happens through the use of system calls.
- Context switch refers to the process of storing a process or a thread such that it can be resumed execution at a later time.
- Context switching happens only in Kernel mode.

(d)

Pros:

- High security and reliability due to most services running in user mode.
- When compared to a monolithic kernel, it is easier to port from one hardware architecture to another.
- It is easy to extend the operating system, since new services are created in the user space and do not require modification of the kernel.
- Architecture is smaller then a monolithic kernel.

Cons:

- Sometimes slow performance due to high system-function overhead.

- Overhead often involved when switching between processes and copying messages.

2.
(a)

```
Output 1 :     Output 2 :     Output 3 :
         0              0              0
         2              1              2
         1              2
```

*Note: *Output* 3 is the result if fork() fails.

(b)

```
#define OUTPUT printf(”%d\n”, i)

main() {
     int i = 0; OUTPUT;

     if (fork()) {
          wait(NULL);
          i += 2; OUTPUT;
     } else {
          i += 1; OUTPUT; return(0);
     }
}
```

3.
(a)
Feasible: A process can go from running to blocked when wait() is called or input is received.

(b)
Not feasible: A process can not go from a blocked to a running state. The process must be in the ready state before it can be in the running state.

(c)
Feasible: A process can go from a blocked to ready state if I/O or event is completed whilst in the blocked state.

(d)
Not feasible: A process in the ready state cannot be blocked because it does not have access to the CPU. The process cannot be blocked if it has not started execution.

(e)
Feasible: A process can go from a ready to running state when the scheduler dispatcher chooses the process for execution.

(f)
Feasible: A process can go from a running to ready state when an interrupt is thrown, which pauses the current process execution.