·TA responsible for HW1: Yilun Jin, yjinas@cse.ust.hk

Fall 2022 COMP 3511 Homework Assignment 1 (HW1)

Handout Date: September 21, 2022, Due Date: October 5, 2022

Name	
Student ID	
ITSC email	@connect.ust.hk

Please read the following instructions carefully before answering the questions:

- You must finish the homework assignment **individually**.
- This homework assignment contains **three** parts: (1) multiple choices, (2) short answer 3) programs with fork()
- **Homework Submission:** Please submit your homework to **Homework #1** on **Canvas**.
- TA responsible for HW1: Yilun Jin, yjinas@cse.ust.hk

1. [30 points] Multiple Choices

Write your answers in the boxes below:

MC1	MC2	MC3	MC4	MC5	MC6	MC7	MC8	MC9	MC10

1) Which of the following components is NOT part of an operating system?
A) The kernel
B) System programs
C) User programs
D) Middleware

2) Which of the following statement about *interrupt* is INCORRECT?
A) Interrupts are used in modern operating systems to handle asynchronous events.
B) Interrupts can only be generated by I/O devices.
C) I/O device can trigger interrupts by sending a signal to the CPU.
D) Interrupts can be caused by software such as trap and exception

3) Which of the following statement is TRUE for *direct memory access* or DMA?
A) DMA transfer data between an I/O device and memory directly
B) A DMA controller requires initialization by CPU
C) DMA frees up CPU from data movement between an I/O device and memory
D) All of the above

4) Which of the following statement is NOT true about the system-call interface?

·TA responsible for HW1: Yilun Jin, yjinas@cse.ust.hk

A) The system call interface enables programs to request services from operating systems
B) The system-call interface intercepts function calls in APIs and invokes the necessary system calls within an operating system
C) The standard C library or `libc` is the system-call interface in UNIX/Linux systems
D) The system-call interface is part of the run-time environment (RTE)

5) Which of the following statement is NOT true in operating system design?
A) A monolithic OS has no structure, but runs efficiently in a single address space
B) A microkernel approach keeps the minimal functionalities in the kernel, thus the operating system has the best overall performance
C) A layered design provides certain level of modularity, which eases the OS design and implementation
D) A loadable kernel module approach offers the flexibility that functional modules can be added and removed during runtime

6) Which of the following statement is CORRECT about *linker* and *loader*?
A) The *linker* combines object modules into a single binary executable file stored on disk
B) The *loader* is responsible to load the executable file into memory
C) The *loader* can also support *dynamically linked libraries* (*DLLs*), which avoids duplication and enables sharing *DLLs* among multiple processes
D) All of the above

7) Which of the following statement is TRUE regarding various scheduling mechanisms used in a mainframe computer?
A) The *long-term scheduler* selects jobs from a job queue and brings them into the memory by allocating resources, which determines the degree of multiprogramming
B) The *medium-term scheduler* swaps some partially executed processes from the memory out to the secondary storage temporarily, which reduces the degree of multiprogramming
C) The *short-term scheduler* selects a process from the ready queue to run on CPU next
D) All of the above

8) _____ occurs when CPU switches from running one process to another.
A) DMA
B) Interrupt
C) Context switch
D) Trap

9) Which of the following statement on `fork()` is NOT true?
A) `fork()` does not require any parameter
B) Once `fork()` is successful, the parent process must wait for the completion of the child process
C) The child process duplicates the entire address space of the parent process
D) `fork()` generates two return values, one for the parent process and one for the child process

·TA responsible for HW1: Yilun Jin, yjinas@cse.ust.hk

10) Which of the following statement is NOT true during process termination in Unix
A) A process terminates by calling `exit()` explicitly or implicitly, which will release all the resources allocated to the process by an OS
B) A terminating process transitions into a *zombie* process temporarily
C) A process can be terminated by its parent process
D) Only after its parent process executes `wait()`, all resources allocated to a terminated process can be fully recovered

2. [30 points] Short answer

(1) (5 points) Please briefly explain what *multiprogramming* and *multitasking* refer to in an OS.

(2) (5 points) Please briefly explain the two essential properties (i.e., *spatial and temporal locality*) why caching works.

(3) (5 points) What are the advantages of separating API and the underling system calls? What is the use of the system call interface?

(4) (5 points) Using *Darwin* , the kernel environment used in iOS and MacOS as an example, please highlight why this is a hybrid design.

(5) (5 points) A process is represented by a *thread* (or multiple threads) and an *address space*. Please illustrate what is contained in threads and the address space, respectively.

(6) (5 points) What is the main purpose of *dual-mode operation*? Can this be extended to more than two modes of operation?

3. (40 points) Simple C programs on fork()

For all the C programs, you can assume that necessary header files are included

1) (10 points) Consider the following code segments:

·TA responsible for HW1: Yilun Jin, yjinas@cse.ust.hk

```
int main()
{
    int i = 0;
    int cnt = 10;
    for (; i<3; i++){
        if (fork() == 0)
            cnt += 10;
        printf("+\n");
        printf(" %d\n ",cnt);
   }
   return 0;
}
```

a) How many '+' will this code print? Please elaborate.

b) How many '20 ' will this code print? Please elaborate.

2) (15 points) Consider the following code segments:

```
int main(){
   pid_t pid = fork();
   int cnt = 0;
   if (pid == 0) {
      cnt += 10;
      pid = fork();
      if (pid != 0) {
         cnt += 10;
         pid = fork();
      }
   }
   printf("%d \n", pid);
   printf("%d \n", cnt);
   return 0;
}
```

a) How many times will this code print none zero process ID (pid)? Please elaborate.

b) How many '0' will this code print? Please elaborate. (Note that we do not consider those 0's, if any, that are contained in non-zero numbers.)

c) How many '10' will this code print? Please elaborate.

3) (5 points) Consider the following code segments:

```
int main()
{
   if (fork() && fork())
       fork();
   printf("1 ");
   return 0;
}
```

How many 1's are printed? Please elaborate.

4) (10 points) Fill in the missing blanks so that the following program will always display the following output:

```
The value x in process 3 is 1
The value x in process 2 is 1
The value x in process 1 is 1
```

Question:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
// fflush(stdout): ensure the output is printed on the
console
int main() {
    int x = 0;
    if (BLANK1) {
```

```
        x = x + 1;
        BLANK2;
        printf("The value x in process 1 is %d\n", x);
        fflush(stdout);
    } else if ( BLANK3 ) {
        x = x + 1;
        BLANK4;
        printf("The value x in process 2 is %d\n", x);
        fflush(stdout);
    } else {
        x = x + 1;
        printf("The value x in process 3 is %d\n", x);
        fflush(stdout);
    }
    return 0;
}
```

BLANK1	
BLANK2	
BLANK3	
BLANK4	