·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	**Casper Kristiansson**
Student ID	**20938643**
ITSC email	**cok@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
C	B	D	C	B	D	D	C	B	D

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.

Multiprogramming is the process of organizing jobs that the CPU executes (code, data). This means that there exist jobs in the memory so that the CPU has always something to execute. For example, if a job requires an IO input the OS knows that it can switch to execute another job while the other is waiting. Doing this will help the CPU to increase its utilization.

Multitasking refers to the processes which allow a user to interact with multiple programs at once. This works by the OS switching between different jobs that the CPU is working on depending on what the user is currently using. It does this using different methods like CPU scheduling if several jobs are running at once and swapping if the job doesn't fit in the memory. An important part of swapping jobs is that they can quickly be done so that the user doesn't have to wait.

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

Locality is an important part of caching. The two basic types are spatial and temporal locality.

Spatial locality builds on the principle that instructions that are stored near a recently executed instruction have a much more likelihood to be executed as well in the near future. **Temporal locality** refers to the principle that an instruction that has been executed has a higher probability to be executed again.

Using these two basic types of the principle of locality plays a big part in why caching works. This means that for both cases the instructions will be cached in the memory and if the system tries to fetch a new instruction there is a high probability that it exists in the cache.

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

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

What is the use of the system call interface?

The **API** is used for providing functions which is available for an application programmer to use. These functions are often well documented and easy to use. The API function will then typically invoke the system calls. This means that the advantage of implementing an API for the OS is to make it easier for developers to develop applications. Because in many cases the system calls could be complex and hard to work with compared to the API.

Another advantage is that if an application is using a specific API the application is expected to run on any system that supports that API.

The **system call interface** is provided by the RTE to serve as a link to the system calls of the operating system. This means that the system call interface intercepts API calls to execute the correct system calls and returns any return values if they exist.

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

The operating system **Darwin** uses the kernel type Hybrid which consists of the BSD Unix kernel and the Mach Microkernel. The goal of a hybrid kernel is to combine benefits from a microkernel and monolithic kernel OS which in case Darwin does.

(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.

The **thread** (or multiple threads) in a process describes the program state. This means that it contains the stack, registers, program counter, and execution flags. A process could contain multiple threads which run in the same address space. Each thread also contains a TCB. The thread control block contains the execution state, scheduling info (state, priority, CPU time), and a pointer to the enclosing process (which process it originated from).

When a program is going to be executed it gets assigned to an **address space** that is different from the main memory. This means that the address space contains the information to run a program such as the instructions to run the program (code) and the programs data.

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

The main purpose of a **dual-mode operation** is to protect the operating system and other system components. There exist two different modes, User and Kernel. The OS uses a mode bit which shows if the code being executed is user code or kernel code. If a privileged instruction tries to be executed in user mode it will be considered illegal and won't be executed.

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

There are cases for example Intel which uses four separate protections modes (rings). The two added operation modes are used for different types of OS services.

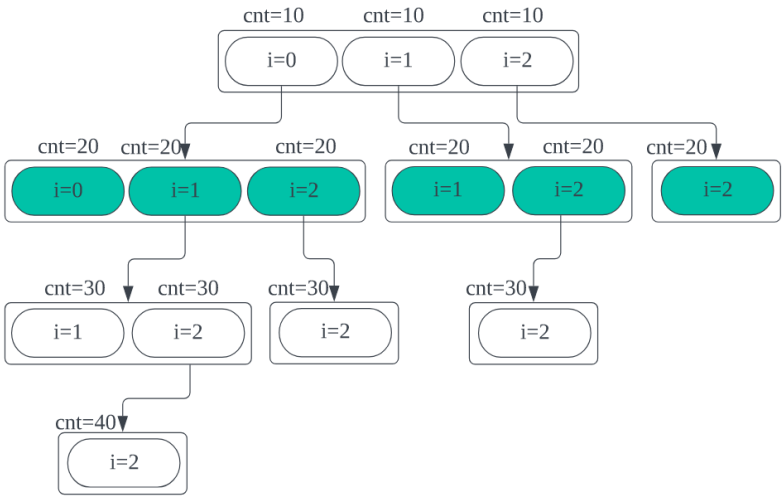
3. (40 points) Simple C programs on fork()

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

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

```
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.



The "+" will be printed 14 times. This can be seen in the diagram above. The parent will create three children. Depending on how much "i" is when it does it the children will create more children. The graph shows exactly how many times the for loop is being run through, meaning the number of times "+" is printed.

Answer: 14

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

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

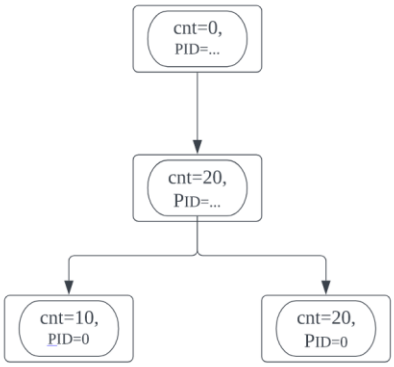
The "20" will be printed 6 times. The diagram contains exactly what each for loop iteration will print. We can see that there are a total of 6 occasions where "20" is being printed. Each time a child is generated the child will have the cnt value increased by 10.

Answer: 6

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.



The program will print a total of two non-zero process IDs. The processes that print the non-zero ID is the parent and the first child. This is because after the first child creates a new child it will not enter the second if statement and just print the return value of fork(). Because it is a child it will be 0. The first child

will then enter the second if statement and create another child. That child will then also print a zero process ID.
Answer: 2

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.)

There will be a total of three "0" printed. This is because two of the IDs printed by the children will be "0" and then the parent cnt value will also print "0".
Answer: 3

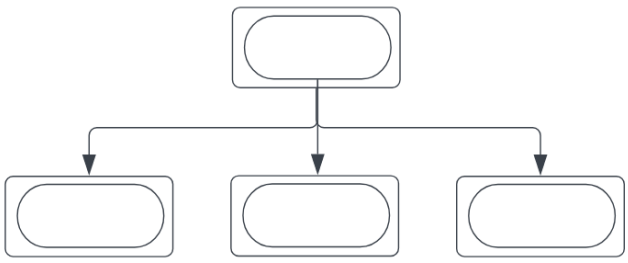
c) How many '10' will this code print? Please elaborate.

The value "10" will only be printed once which is the cnt value for the second child created. This can easily be seen because after the first child creates the next one the "pid" value will be zero which means that it will directly print the cnt value which in this case is "10".
Answer: 1

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.



There will be a total of four "1 " printed. This is because the parent process will create three children. Because the process id is returned to the parent it will enter the if statement. The first child will not enter or initiate the second fork due to the and comparison already yielding false. Therefore, there will be a total of four processes and a total of four "1" (no children perform the fork() command).
Answer: 4

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	fork()!=0

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

BLANK2	`wait(NULL)`
BLANK3	`fork()!=0`
BLANK4	`wait(NULL)`