Fall 2023 COMP 3511 Homework Assignment #1

Handout Date: September 18 (Monday), 2023, Due Date: October 2 (Monday), 2023

Name	**Chloe Hu**
Student ID	**21044009**
ITSC email	chuap@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: Haoxuan Yu, hyubc@connect.ust.hk. Please contact the TA if you have any question about this homework.

1. [30 points] Multiple Choices

Write your answers in the boxes below:

MC1	MC2	MC3	MC4	MC5	MC6	MC7	MC8	MC9	MC10
D	C	B	A	D	D	C	D	B	D

1) Which of the following statements about I/O operations is true?

A) Programmable I/O allows CPU to run special I/O instructions to move data between memory and I/O devices

B) DMA controller can move data between memory and an I/O device

C) I/O devices send *interrupt* to CPU when requiring CPU attention

D) All of the above

2) Which of the following statements about *interrupt* is NOT true?

A) Interrupts are used to handle asynchronous events

B) Interrupts can have different priorities

C) Interrupts can only be generated by I/O devices.

D) Interrupts can be caused by software such as *trap* and *exception*

3) Which of the following statements is NOT true for *virtualization*?

A) It allows an OS to run as an application within other OS

B) Applications can run faster on a virtual machine than on a physical machine

C) It enables multiple OSes to run on a single physical machine

D) It provides better isolation in that the corruption of one virtual machine will not crash the entire system

4) Which of the following statements is NOT true about secondary storage?
A) Nonvolatile memory (NVM) devices are cheaper than hard-disk drives (HDDs)
B) NVM devices run faster than HDDs
C) NVM devices are more reliable than HDDs
D) NVM devices usually have smaller capacity than HDDs

5) Which of the following statements is true on operating system design?
A) A monolithic OS runs more efficiently in a single address space
B) A microkernel design maintains minimal functionalities in the kernel, thus it can be more easily ported to different hardware platforms
C) The modern OS does not rely on a single design approach, rather adopts a hybrid approach combining different type of design for performance, security, and usability
D) All of the above

6) Which of the following statements is NOT true about the separation of kernel mode and user mode?
A) It provides basic means for protection
B) Certain privileged instructions could only be executed when the CPU is executing in the kernel mode
C) This allows user programs and kernel programs to execute more efficiently
D) It does not allow hardware devices to be accessed when executing in the user mode.

7) Consider the following C program that uses the fork() system call:

```
#include <stdio.h>
#include <unistd.h>

int main() {
    if (fork() || fork())
        printf("1\n");
    return 0;
}
```

How many 1s are printed?
A) 1
B) 2
C) 3
D) 4

8) Which of the following steps are necessary in fork()?
A) Create and initialize PCB
B) Allocate memory and duplicate the entire contents of the address space of the parent
C) Inherit the execution context of the parent, e.g., any open files
D) All of the above

9) Which of the following elements is NOT distinctive between a parent process and its child process right after fork() system call?
A) process ID or pid of the process
B) Program counter or PC value
C) The return value from fork()
D) Process running state

10) Which of the following statements is NOT true about named pipe
A) A named pipe ceases to exist after the communicating processes have terminated
B) A named pipe does not require parent-child relationship for communicating processes
C) A named pipe allows multiple (more than two) processes to use it for communication
D) A named pipe is bidirectional – allowing communications in both directions

2. [30 points] Please answer the following questions in a few (short) sentences

(1) (5 points) Please briefly explain why a *multitasking* or *time-sharing* system is considered a natural extension of a *multiprogramming* system? Which one is better suited for interactive applications?

In multiprogramming, the OS loads several jobs into memory and switches rapidly between them. The OS will select a process that is ready to run. If the process needs to wait for input/output, other processes will run, improving CPU utilisation.

Multitasking/timesharing can be considered an extension of a multiprogramming system since it uses CPU scheduling and time-slicing to rapidly switch the CPU between different processes. CPU scheduling decides which jobs should be performed first. Time-slicing gives each job a short period of time to execute before moving on to a different job, giving the illusion of concurrent execution. Timesharing also utilises virtual memory which allows the execution of processes not completely in memory, which reduces the overhead associated with swapping data in and out of the main memory.

As a result, multitasking/timesharing is more suited for interactive applications as it provides rapid response times to the user, allowing users to feel like they are executing jobs in real time.

(2) (5 points) Please briefly explain why *caching* would work because of the locality.

The two types of locality, temporal and spatial. Temporal locality stores items which are recently accessed. Spatial locality stores items that are spatially close to each other ie. contiguous blocks of data. Caching works due to locality as it allows the storage of data that is frequently accessed, which in turn means that frequently accessed data can be quickly accessed by the user.

(3) (5 points) Please illustrate how the four-tuple (i.e., client port number, client IP address, server port number and server IP address) are uniquely determined when a client initiates a connection with a server

Server IP address and port number are publicly known. When a client process initiates a request for a connection, it selects a client port number (greater than 1024), that isn't currently used and the client IP address to identify itself on the network. Meanwhile, the server listens for incoming connections from that specific client socket.

(4) (5 points) Please illustrate the **three** advantages of multiprocessor systems and **two** advantages of multicore systems over conventional multiprocessor systems.

Multiprocessor systems have:

- Increased throughput: Multiple processors can run multiple tasks at a single time meaning more computing capability
- Economic scalability: Multiprocessor systems can be scaled up by adding more multiprocessors allowing for increased computing capability if workload demands increase. Furthermore, one multiprocessor system can cost less than equivalent multiple single processor systems, since they can share peripherals, mass storage, and power supplies.
- Increased reliability: Since there are multiple processors, if one processor fails the other functioning processors continue to execute tasks, giving sufficient time to fix the issue. Thus, the system has a degree of fault tolerance.

Multicore systems have:

- Fast communication: Since there are multiple cores residing on a single chip, communication between different cores on the chip is faster than communication between chips.
- Uses significantly less power: As multicore systems have multiple cores on a single chip, it uses significantly less power compared to single core systems as it can share resources and operate more efficiently. As a result, it is important for mobile devices.

(5) (5 points) In OS design and implementation, what do we refer to as *policy* and *mechanism*, respectively? Please illustrate with a concrete example why it is necessary to separate *policy* and *mechanism*.

The policy is "what will be done". It refers to the rules that determines how a feature

should behave. While the mechanism is "how to do it", which refers to the methods used to achieve the policy.

It is necessary to separate the two to create a flexible and modular system where modifications are easier to make. For example: A file system that allows you to read a file but not write it. The file system is the mechanism. The policy is read-only. Since the two are separated, if we wanted to modify the system, we could change the policy to be read and write without changing the mechanism.

(6) (5 points) Please describe the necessity steps that a parent can use a *pipe* to **write** to a child process.

1. Create a pipe using pipe()
2. Create a child process using fork()
3. Close the appropriate ends of the pipe. The parent closes the *read* end and the child closes the *write* end.
4. In the parent process, write data to the pipe using write()
5. In the child process, read data from the pipe using read()

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 sample codes from a C program. Notes: executables cmd1 and cmd2 do not generate outputs.

```
if(fork() == 0) {
  exec(cmd1);
  printf("1");
}
else {
  if(fork() == 0) {
    exec(cmd2);
  }
  else {
    wait(NULL);
    wait(NULL);
  }
  printf("2");
}
printf("3");
```

```
if(fork() == 0) {
  exec(cmd1);
  printf("1");
}
else {
  wait(NULL);
  if(fork() == 0) {
    exec(cmd2);
  }
  else {
    wait(NULL);
    printf("2");
  }
}
printf("3");
```

(a) In the code shown above, do the two commands cmd1 and cmd2 execute serially (one after the other) or in parallel? (2 pts) ANS: parallel

(b) How many "1", "2", "3" are there in the outputs respectively? (3 pts) ANS: one 1, two

2's, three 3's

(c) Indicate how you would modify the code above to change the mode of execution from serial to parallel or vice versa. That is, if you answered "serial" in part (a), then you must change the code to execute the commands in parallel, and vice versa. Indicate your changes next to the code snippet above. (5 pts)

Since the execution of cmd1 and cmd2 is parallel, to change it to serial we would have to utilise wait() in the first else statement to ensure that the parent process waits for the first child to complete before calling the second fork(). As a result, the parent process would wait for the first child to call cmd1 before forking and calling cmd2.

2) (10 points) Suppose the following C code segment is executed. How many letters will be printed? How many 'a's, 'b's and 'c's will be printed on the screen with this code segment? Explain briefly.
(Hint: The letter will first be placed at the buffer after the "printf()" instruction, then be printed to the screen after the "return" operation. And a child process will copy the buffer of her parent process.)

```
int index = 0;
for(int i = 0; i < 2; i++){
    int pid = fork();
    if(pid != 0){
        fork();
        printf("a");
    }
    else if(index == 0){
        fork();
        index++;
        printf("b");
    }
    else{
        fork();
        printf("c");
    }
}
```

ANSWER: 16 a's, 12 b's, 4 c's

Firstly, the code calls fork() which doubles the number of processes each iteration. In the first iteration, a parent process forks creating a child, both of which print 'a'. The child process forks creating another child, both of which print 'b'. Since fork was called initially, doubling the processes we have 4 a's and 4 b's.

In the second iteration the amount of a's are squared, leaving us with 16 a's. 8 more b's are printed since only the children print 'b' and the remaining processes fork to print 4 c's.

3) (10 points) What is the output of the following program? Please explain your answer.

```
void print_message(const char* msg) {
    printf("%s\n", msg);
    fflush(stdout);
}
int main() {
    if ( fork() ) {
        wait(0);
        if ( fork() ) {
            wait(0);
            print_message("A");
        } else {
            print_message("B");
        }
    } else {
        if ( fork() ) {
            wait(0);
            print_message("C");
        } else {
            print_message("D");
        }
    }
    return 0;
}
```

ANSWER: DCBA

The code forks and the parent process enters the first IF statement. It wait() is called, thus the process must wait for the child process to complete first. The child process occurs when fork() == 0 which is triggered by the ELSE statement. In the else statement we see a similar pattern, where a fork is called in the if statement. The parent has to wait for the else statement to finish outputting before starting. As a result the first letter to be printed is 'D', followed by the 'C'.

Since the child of the first fork() has finished outputting we return back to the first IF statement. Here we see a similar pattern again where the parent is made to wait for the child to finish first. As a result we will see 'B' printed followed by 'A'.

4) (10 points) The following question is related to the PA1 shell project this semester. For details, please refer to the corresponding project-related lab. Consider the following C program:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <signal.h>
void handler1(int sig) {
    printf("Handler1\n");
    exit(1);
}
void handler2(int sig) {
    printf("Handler2\n");
}
int main() {
    char text[100];
    if ( fork() ) {
        signal(SIGINT, handler2);
        wait(0);
        while (1) {
            printf("Parent> ");
            scanf("%s", text);
            if (strcmp(text, "exit") == 0) exit(0);
        }
    } else {
        signal(SIGINT, handler1);
        while (1) {
            printf("Child> ");
            scanf("%s", text);
            if (strcmp(text, "exit") == 0) exit(0);
        }
    }
    return 0;
}
```

(a) In the child process, what happens if Ctrl-C is pressed when entering a text message? Will the child process terminate and return to the parent process?
If Ctrl-C is pressed when entering a text message, the child processes will terminate due to the exit(1) in the handler1 function and return to the parent process. This is because exit() terminates the process.

(b) In the parent process, what happens if Ctrl-C is pressed when entering a text message? Will the parent process terminate and return to the system shell?

If Ctrl-C is pressed in the parent process, the process will not return to the system shell since there is no exit() within the handler2 function. The while loop will continue to run in an infinite loop.