TA responsible for HW1: Jingcan Chen (jchenhv@cse.ust.hk)

Fall 2024 COMP 3511 Homework Assignment #1

Handout Date: September 15, 2024, Due Date: October 2, 2024

Name	LI,Yuntong
Student ID	20944800
ITSC email	ylino@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: Jingcan Chen,jchenhv@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	D	B	B	C	A	D	D	C	B

1) Which of the following services is NOT necessary for an operating system?

A) File System
B) Error Detection
C) GUI
D) I/O Operation

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

A) Interrupts are used to handle asynchronous events, e.g., I/O ops and software errors
B) Interrupts can have different importance specified by priority levels
C) Upon each interrupt, a piece of OS codes is called and executed
D) Interrupts cannot be generated by external hardware

3) Which of the following statements is **NOT true** on *process* and *thread*?

A) At a certain moment, only one process is running on a CPU core
B) One process with a single thread can run on different CPU cores at a time in parallel
C) A process can consist of multiple thread(s), and they run within the same address space
D) One process consists of only one address space

TA responsible for HW1: Jingcan Chen (jchenhv@cse.ust.hk)

4) Which of the following statements is **not true** about the goal of *operating system*?
(A)Execute user programs and make solving user problems easier
(B)Allow or deny user's access to hardware resources
(C)Make the computer system convenient to use
(D)Manage and use the computer hardware in an efficient manner

5) Which of the following statements is **true** about *system calls* and *dual mode operation*?
A) Under dual mode operation, both user and the operating system can access all resources
B) The concept of dual modes can only have two modes
C) Some instructions are privileged in system calls
D) Context switch happens during system calls

6) Which of the following statements is **not true** about *loadable kernel module* approach?
A) All components in the kernel can be dynamically added and removed during runtime
B) The kernel has a set of core components and can linkin additional services via modules, either at boot time or during run time.
C) Module approach resembles a layered design in that each kernel section has a well-defined, protected interface, and it is flexible to call other modules
D) Loadable modules can add new services to the kernel without recompiling the whole kernel.

7) 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 without any assistance from CPU
B) DMA completes a data transfer without interrupting CPU
C) DMA frees up CPU from data movement between an I/O device and memory
D) All of the above

8) Which is true about *signal handling* in operating systems?
A) Signals cannot be ignored and should terminate the program since they hurt system performance
B) Keyboard interrupt is a synchronous signal because it happens during the process execution, as the illegal memory access
C) Signals should be sent to every thread in a multi-threaded process
D) Each signal has a default handler function

9) When the degree of multiprogramming is too high, ___
A) Long-term scheduler will remove some processes from the memory
B) Short-term scheduler assigns less CPU cores to a certain process
C) Mid-term scheduler swaps out certain processes to the disk
D) None of the above

TA responsible for HW1: Jingcan Chen (jchenhv@cse.ust.hk)

10) Which is **not true** about *interprocess communication (IPC)*?

A) There are generally two models of IPC, i.e. shared memory and message passing

B) Pipes can be used only in the parent-child processes or among multiple threads of a process

C) Message sending need not block the processes

D) Different processes communicating through sockets must have different port numbers

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

(1) (5 points) Please describe what the CPU needs to specify prior to DMA operations, and illustrate why this is better than programmed I/O when moving large chunks of data.

Before initiating DMA operations, the CPU must specify the source and destination addresses, read or write mode, the number of bytes to be transferred and the controller information.

DMA is preferable to programmed I/O for large data transfers because it offloads the data handling work to the DMA controller, freeing the CPU to execute other tasks and enhancing system efficiency.

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

Spatial locality is the concept that sequential memory locations are often accessed closer together in time.

Temporal locality refers to the idea that a recently accessed memory location is likely to be accessed again.

These two essential properties storing frequently and recently used data closer to CPU, reducing access times to make the caching work.

TA responsible for HW1: Jingcan Chen (jchenhv@cse.ust.hk)

(3) (5 points) What resources are shared and not shared in a multi-threaded process? What is the main benefit of using multiple threads instead of multiple processes?

In a multi-threaded process, threads share all resources such as memory content,address, but they have separate stacks, stack pointers and CPU registers.

Using threads instead of processes offers benefits like reduced overhead in context switching and better resource sharing, making communication within an application faster and more efficient. It takes less time and space to use multi-thread than multi-processes.

(4) (5 points) What are the advantages of providing *system call APIs* to users in *dual-mode system*? Please explain from user view and system view.

For users, It can invoke abstracted system functions, enhancing portability as different hardware platforms can utilize the same APIs. This simplifies coding for users by making low-level system or hardware details less significant. Additionally, user code becomes safer since it's more challenging to crash the system while operating in user mode.

For system, core operations should be accessed exclusively through system calls to manage access and enhance stability. By concealing direct access to critical resources, this approach safeguards the system and protects it from potential damage.

(5) (5 points) What is *orphan* process? What is the problem with an *orphan* process? How does OS handle that?

An orphan process is one whose parent has terminated without invoking wait(), leaving it without supervision.

This situation may forever remain a zombie process, wasting system memory.

The `init` or `systemd` process acts as the parent for orphaned processes. It periodically calls the `wait()` function to collect the exit status of these orphaned processes. This action also releases the process identifiers and clears the corresponding entries in the process table.

(6) (5 points) What is *copy-on-write* ? What is the advantage of using *copy-on-write* in fork() implementation in Linux?

Copy-on-write is a technique in which child processes share the parent's memory pages until a modification is made.

It will have better memory efficiency and it can save the memory. It has a faster process creation time.

3. **(40 points)** Simple C programs on fork(). Suppose all printf() will be followed by fflush(stdout) even if it is not presented in the code. For all the C programs, you can assume that necessary header files are included

1) (5 points) Consider the following code segments:

```
int main()
{
    pid_t pid1;
    pid_t pid2;

    pid1 = fork();
    pid2 = fork();

    printf("pid1:%d, pid2:%d\n", pid1, pid2);
}
```

(a) How many processes are there in total when this code finishes? Give the answer.

(b) If one process prints "pid1:51234, pid2: 51235", one process prints "pid1:0, pid2: 51236", write down other processes' pid and their outputs.

(a) 4 processes

(b) For process with pid 51235, it prints "pid1:51234, pid2:0"
For process with pid 51236, it prints "pid1:0, pid2:0"

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

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

(a) Who will print the "cnt", the child process or parent process? Why? How many times of "printf" will this code execute? Give the answer with explanation.

Only child processes print "cnt" because pid == 0 in the if condition corresponds to the child processes.

7 times of "printf" will this code execute. Because in the first loop, 1 child prints. In the second loop, 2 new children print. In the third loop, 4 new children print. So the total number is 1+2+4 = 7 times.

(b) Theoretically, based on Linux fork() mechanism introduced in the lecture, how many memory copies of the variable "cnt" will there be? Explain your answer.

There will be 8 memory copies of variable "cnt" will there be.

Because each fork() results in the creation of a new process with a separate copy of the memory space, including variables like cnt. So, initially, there is 1 copy. After the first folk(), there are 2 copies. After the second folk(), there are 4 copies. After the third folk(), there are 8 copies.

(c) Directly running this piece of code may produce different outputs every time. Briefly explain why.

Multiprocessing refers to the ability of different processes to run simultaneously, either through context switching or by executing on separate CPU cores. As a result, due to slight variations in real-world conditions, the order of output may vary.

TA responsible for HW1: Jingcan Chen (jchenhv@cse.ust.hk)

3) (10 points) Consider the following code segments:

```
#include <stdio.h>
#include <unistd.h>
int main()  {
   for  (int i = 0; i < 2; ++i)  {
      if  ((fork()  && fork())  ||  !fork())  {
         printf("A");
         fflush(stdout);
       }
   }
}
```

(a) Determine the total number of "A"s output by the given program, assuming it operates normally. Provide a brief explanation for your answer.

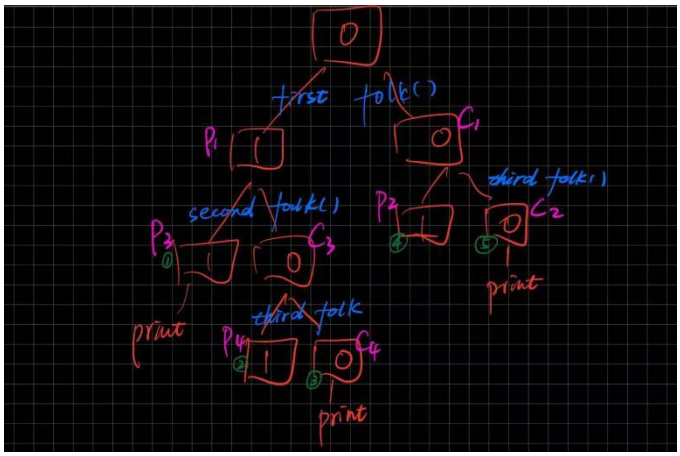
The total number is 18.

Explanation:

1) i = 0
In the if condition, the first fork() will create a child process, we have P1 = 1, C1 = 0.
We first consider the C1, since the C1 = 0, (0 && fork()) is obviously 0, it will not execute the second fork(), the C1 will go to the third fork(), then it becomes the P2 = 1, C2 = 0. For the (0 || !fork()), so C2 will execute the print("A"), P1 will do nothing.
Then we consider the P1, P1 will execute the second fork(), it will create new child process, then it becomes P3 = 1, C3 = 0. For the P3, then it has (1 && P3) = 1, (1 || !fork()) obviously is 1, so it will not execute the third fork(), execute the print("A"). For the C3, (1 && C3) = 0, it will go on the third fork(), then it becomes P4 = 1, C4 = 0. So for the (0 || !fork()), C4 will execute the print("A") and P4 will do nothing.
So for this loop it has 3 print.

2) i = 1
After the first loop, there are 5 process go to this loop(P3,P4,C4,P2,C2) and each will do the same as the first loop, so each will print 3 times. Then for this loop it will have 15 print.

So in total, it will have 3+15 = 18 print("A").



TA responsible for HW1: Jingcan Chen (jchenhv@cse.ust.hk)

4) (10 points) Fill in the missing blanks using "printf" and "wait" functions, so that the following program will always display the following output:

Output: CDBA

Question:

```
int main() {
    if ( fork() ) {
        wait(0);
        if ( fork() ) {
            BLANK1;
            BLANK2;
            fflush(stdout);
        } else {
            printf ("B");
            fflush(stdout);
        }
    } else {
        if ( !fork() ) {
            BLANK3;
            fflush(stdout);
        } else {
            BLANK4;
            printf ("D");
            fflush(stdout);
        }
    }
    return 0;
}
```

BLANK1	wait(0)
BLANK2	printf("A")
BLANK3	printf("C")
BLANK4	wait(0)