Fall 2024 COMP 3511 Homework Assignment #1

Handout Date: September 23, 2024, Due Date: October 7, 2024

Name	**Solution**
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: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	A/C	D	C	B/D

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
Answer: C

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
Answer: D

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
Answer: B

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
Answer: B
ACD from Slide 1.21

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
Answer: C
B in slide 3.22; D context switch happens when changing to another process, but system call of a process does not incur the change of process

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 link in 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.
Answer: A
Slide 2.50 The kernel has a set of core components and can link in additional services via modules, either at boot time or during run time. So the core components cannot be removed.

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
Answer: A/C
CPU needs to specify some metadata for DMA. But A option is a bit vague so it could be right in the sense that "transfer" means during transferring.
For B, Slide 1.43 when transferring is done DMA will send interrupt to CPU for signaling completion. It is incorrect.

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
Answer: D
Slide 4.28-4.30

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
Answer: C
Slide 3. 18

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
Answer: B and D
Slide 3.58

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.

Answer:
It needs to specify (3 points) (some students miss the load/store)
- the addresses of the memory and the target device (source & destination)
- load or store operation (read or write depending on whether input or output device),
- number of bytes transferred

Advantage: This frees up the CPU from executing data movement instructions, so CPU can do more useful things (2 point).

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

Answer: The two essential properties are *temporal locality*, referring to the fact that an item recently referenced or fetched will likely be referenced again in the near future, and *spatial*

locality, referring to the fact the nearby items of a recently accessed item will likely be referenced in the near future.
Cache wors because it improves the hit ratio on the fast cache, reducing the time to carry data from the disk.
Note: It is important to state "why" cache works, instead of just stating what is done in cache. Future reference is the key reason. Failing to answer it results in a 2-point deduction.

(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?

Answer:
Different threads shares with other threads of the same process its code, data, and other OS resources, such as open files and signals (2 points), but they have unique thread IDs, program counter (PC)s, a register set, and a stack. (2 points). Slides 4.6
Process creation is heavy-weight while thread creation is light-weight, in which different threads belonging to the same process share code, data and others. (1 point). Slides 4.4 or anything in 4.7

(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.

Answer:
User: (3 points, 2pt for 1 correct and 3pt for 2)
(1) program portability by using API implying that programs do not need to be altered as long as APIs are the same
(2) to hide the complex details in system calls
System: (2 points)
The dual-mode operation provides the basic means of protecting the operating system from errant users and errant users from one another.

Main idea:
For user: to easily call low-level functions.
For system: anything related to system protection.

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

Answer:
A process becomes an *orphan* process when its parent process terminates without call `wait()`. (1 points)
The problem with this is that parentless terminated processes would forever remain zombies, wasting system memory. (2 points)
The solution is to re-parent the *orphan* process, often assigned to `systemd` process. (2 points)

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

Answer:
COW is a technique to delay or prevent copying of data. Rather than duplicate the process address space, the parent and the child can share a single copy. (2 points)
In the case that the pages are never written — for example, if **exec()** is called immediately after **fork()** — they never need to be copied. This greatly simplifies and speeds up the process creation. (3 points)
(if say "save memory" -1pt)

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) (5points) 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.

Answer:

(a) 4. (2 points)

(b) "pid1:51234, pid2: 0" from **process 51235**, "pid1:0, pid2: 0" from **process 51236** (3pts) (if no process pid, -1)

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.

Answer:

Child process, since the child process has the return value 0 from fork(). (1 point)

7 times. (1 points)

Suppose parent process is p0.

i == 0: p0 -> p0, p1, and p1 will print

i == 1: p0 -> p0, p2; p1 -> p1, p3; p2 and p3 will print

i == 2: p0 -> p0, p4; p1 -> p1, p5; p2 -> p2, p6; p3 -> p3, p7; and p4, p5, p6, p7 will print (3 point)

(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.

Answer:

8 copies (7 new ones and 1 original; answer copy 7 times is also ok) (2 points). Based on the copy-on-write mechanism, when i == 3, the 4 new processes (p4-p7) will only read the "cnt" for printf, but their parents will **write cnt** so that they will copy. (3 points)

(if answer is 4 with COW mechanism, -4; the key is when writing "cnt" the parent will copy. If no, -2; if say "copy when fork", -3; if say "child copy" -1)

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

Answer: The execution order of the parent process and the child process is not guaranteed. Thus, either of them may print first, producing different outputs. (5

points) **(if answer is "parent did not wait for child to end", -3. Even when wait() is called, the output can still be different every time)**

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.

Answer: 18 (2 point). Suppose parent process p0

When i == 0:

for the 1st fork(), **(2 point)**

p0 -> p0, p1

for the 2nd fork(), **(2 point)**

p0 -> p0, p2, and p0 goes to print; p2 continues the "if" condition

p1 does not execute 2nd fork() since the 1st fork() it has 0, so skip the condition statement.

For the 3rd fork(), **(2 point)**

p1 -> p1, p3, and p1 fail the "if" and skip; p3 goes to print

p2 -> p2, p4, p2 fails the "if" and skip; p4 goes to print

For i == 1, each of the 5 processes executes as above. **(2 point)**

In total there are 3 + 3 * 5 = 18 "A"s.

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	
BLANK2	
BLANK3	
BLANK4	

Answer:

```
int main() {
    if ( fork() ) {
        wait(0);
        if ( fork() ) {
            wait(0);
            printf("A");
            fflush(stdout);
        } else {
            printf("B");
            fflush(stdout);
        }
    } else {
        if ( !fork() ) {
            printf("C");
            fflush(stdout);
        } else {
            wait(0);
            printf("D");
            fflush(stdout);
        }
```

```
    }
    return 0;
}
```

Note:
One blank = 3 points.
Two blanks = 5 points.
Three blanks = 8 points.
Four blanks = 10 points.

wait(NULL) is also ok. If you put wait() in BLANKs 1 and 4, a total of 2 points will be deducted.

Another answer for blank 3 and 4 (should be presented together):
BLANK3: printf(””)
BLANK4: printf(“C”)