United International University
Course: Operating Systems (CSE 4509)
Course Teacher: Azizur Rahman Anik
Topics: Process, fork, wait, exec

1) Consider a parent process P that has forked a child process C. Now, P terminates while C is still running. Answer yes/no, and provide a brief explanation.
   a) Will C immediately become a zombie?
   b) Will P immediately become a zombie, until reaped by its parent?

   Ans:
   (a) No, it will be adopted by init.
   (b) Yes.

2) Consider a process P in xv6 that invokes the wait system call. Which of the following statements is/are true?
   a) If P does not have any zombie children, then the wait system call returns immediately.
   b) The wait system call always blocks process P and leads to a context switch.
   c) If P has exactly one child process, and that child has not yet terminated, then the wait system call will cause process P to block.
   d) If P has two or more zombie children, then the wait system call reaps all the zombie children of P and returns immediately.

   Ans: (c)

3) Consider a process P that executes the fork system call twice. That is, it runs code like this: int ret1 = fork(); int ret2 = fork(); How many direct children of P (i.e., processes whose parent is P) and how many other descendants of P (i.e., processes who are not direct children of P, but whose grandparent or great grandparent or some such ancestor is P) are created by the above lines of code? You may assume that all fork system calls succeed.
   a) Two direct children of P are created.
   b) Four direct children of P are created.
   c) No other descendant of P is created.
   d) One other descendant of P is created.

   Ans: (a), (d)

4) Consider the following C program. Assume there are no syntax errors and the program executes correctly. Assume the fork system calls succeed. What is the output printed to the screen when we execute the below program?

```
void main(argc, argv) {

  for(int i = 0; i < 4; i++) {
    int ret = fork();
    if(ret == 0)
      printf("child %d\n", i);
  }
}
```

   Ans: The statement “child i” is printed 2 i times for i=0 to 3

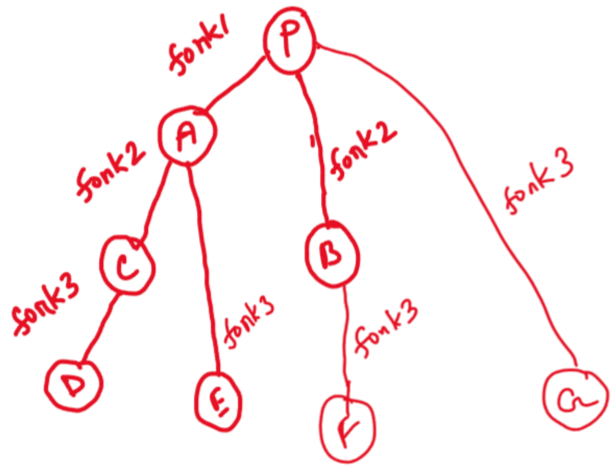
5)

Consider the following pseudocode. Assume all system calls succeed and there are no other errors in the code.

```
int ret1 = fork(); //fork1
int ret2 = fork(); //fork2
int ret3 = fork(); //fork3
wait();
wait();
wait();
```

Draw the process tree for the above code.

Ans:



6)

Consider a parent process that has forked a child in the code snippet below.

```
int count = 0;
ret = fork();
if(ret == 0)  {
printf("count in child=%d\n", count);
}
else  {
count = 1;
}
```

The parent executes the statement "count = 1" before the child executes for the first time. Now, what is the value of count printed by the code above? Assume that the OS implements a simple fork (not a copy-on-write fork).

Ans: 0 (the child has its own copy of the variable)

7) Consider a process P1 that forks P2, P2 forks P3, and P3 forks P4. P1 and P2 continue to execute while P3 terminates. Now, when P4 terminates, which process must wait for and reap P4?

Ans: init (orphan processes are reaped by init)

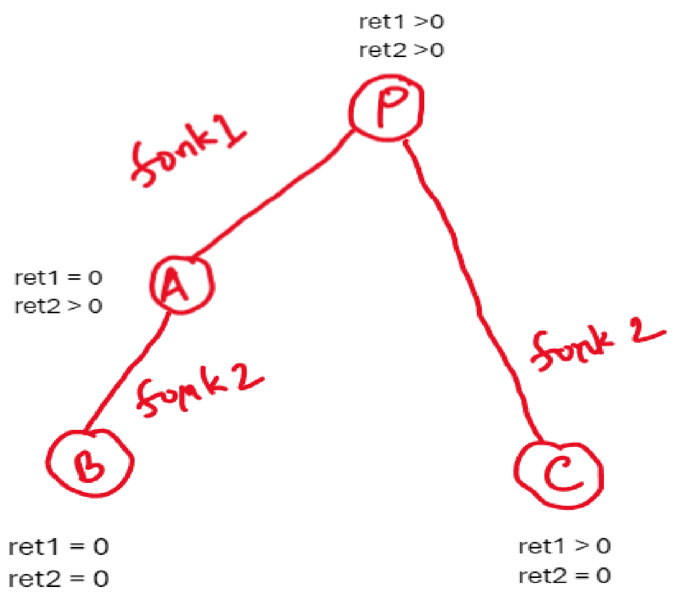
8)

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
void main(){
    pid_t ret1 = fork();
    pid_t ret2 = fork();

    if(ret2 == 0) {
        printf("Os is the worst!\n");
    }
    else if (ret1 == 0 && ret2 > 0) {
        wait(NULL);
        printf("OS is okay!\n");
    }
    else if (ret1 > 0 && ret2 == 0){
        wait(NULL);
        printf("OS is fun!\n");
    }
    else if (ret1 > 0 && ret2 > 0){
        wait(NULL);
        printf("OS is confusing!\n");
    }
    else {
        wait(NULL);
        printf("OS is the best!\n");
    }
}
```

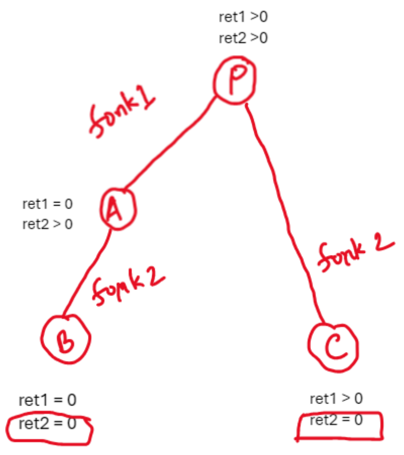
Draw the process tree and show possible output.

Ans:



```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
void main(){
      pid_t ret1 = fork();    → fork1
      pid_t ret2 = fork();    → fork 2

      if(ret2 == 0) {   → B, C
            printf("Os is the worst!\n");
      }
      else if (ret1 == 0 && ret2 > 0) {   → A
            wait(NULL);   → will wait for child B
            printf("OS is okay!\n");
      }
      else if (ret1 > 0 && ret2 == 0){   → C
            wait(NULL);
            printf("OS is fun!\n");
      }
      else if (ret1 > 0 && ret2 > 0){   → ?
            wait(NULL);
            printf("OS is confusing!\n");
      }
      else {
            wait(NULL);
            printf("OS is the best!\n");
      }
}
```



Outputs:

Os is the worst!
Os is okay!
Os is the worst!
OS is confusing!

9)

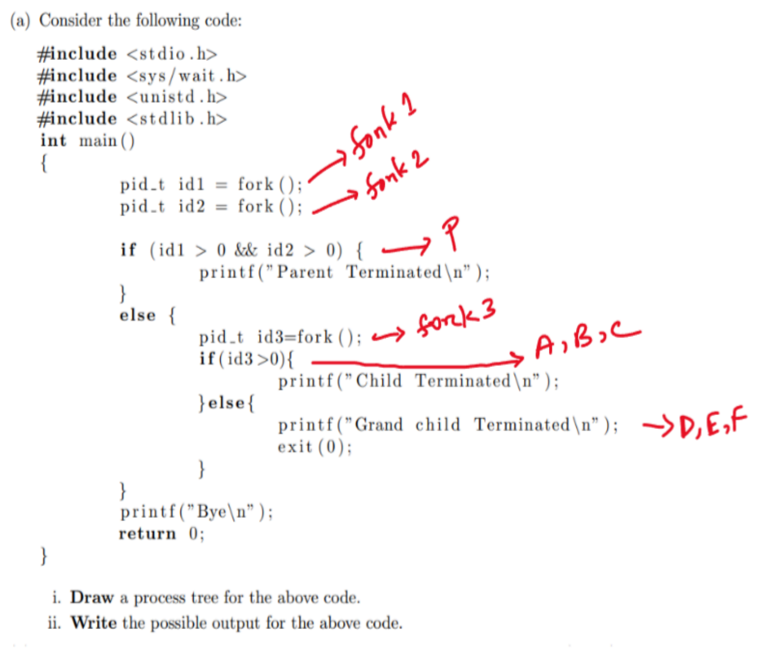
(a) Consider the following code:

```
#include <stdio.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdlib.h>
int main()
{
        pid_t id1 = fork();
        pid_t id2 = fork();

        if (id1 > 0 && id2 > 0) {
                printf("Parent Terminated\n");
        }
        else {
                pid_t id3=fork();
                if(id3>0){
                        printf("Child Terminated\n");
                }else{
                        printf("Grand child Terminated\n");
                        exit(0);
                }
        }
        printf("Bye\n");
        return 0;
}
```

i. **Draw** a process tree for the above code.
ii. **Write** the possible output for the above code.

Solution:



(a) Consider the following code:

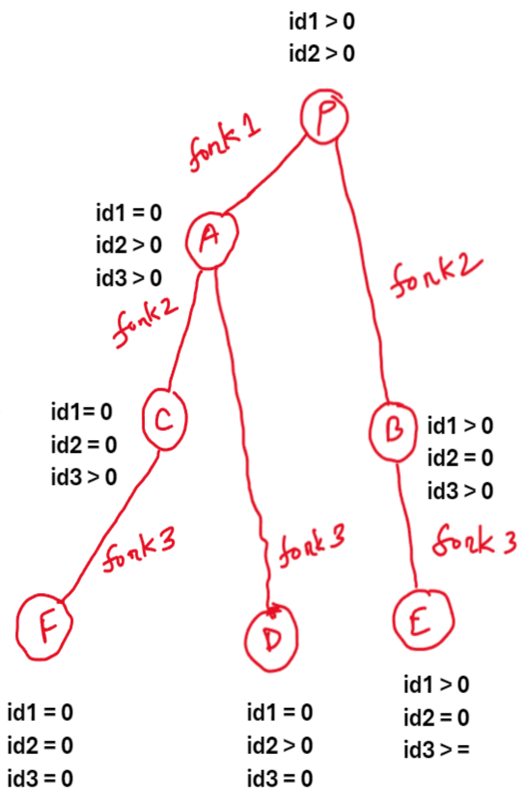
```
#include <stdio.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdlib.h>
int main()
{
        pid_t id1 = fork();
        pid_t id2 = fork();

        if (id1 > 0 && id2 > 0) {
                printf("Parent Terminated\n");
        }
        else {
                pid_t id3=fork();
                if(id3>0){
                        printf("Child Terminated\n");
                }else{
                        printf("Grand child Terminated\n");
                        exit(0);
                }
        }
        printf("Bye\n");
        return 0;
}
```

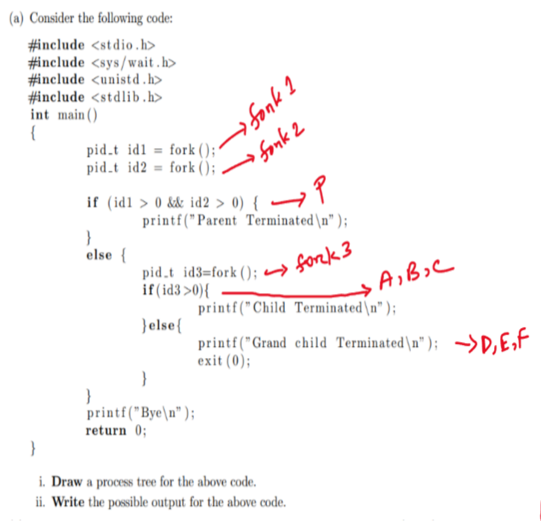
i. **Draw** a process tree for the above code.

ii. **Write** the possible output for the above code.

fork 1
fork 2
P
fork3
A, B, C
D, E, F



Parent Terminated
Bye
Child Terminated
Grandchild Terminated
Bye
Bye
Child Terminated
Grandchild Terminated
Bye
Bye
Child Terminated
Grandchild Terminated
Bye
Bye



(a) Consider the following code:

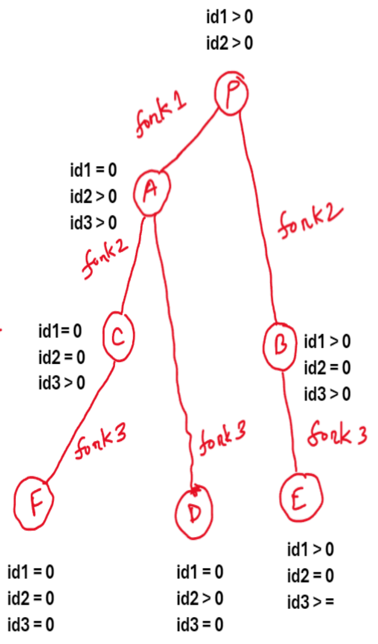
```
#include <stdio.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdlib.h>
int main()
{
        pid_t id1 = fork();
        pid_t id2 = fork();

        if (id1 > 0 && id2 > 0) {
                printf("Parent Terminated\n");
        }
        else {
                pid_t id3=fork();
                if(id3>0){
                        printf("Child Terminated\n");
                }else{
                        printf("Grand child Terminated\n");
                        exit(0);
                }
        }
        printf("Bye\n");
        return 0;
}
```

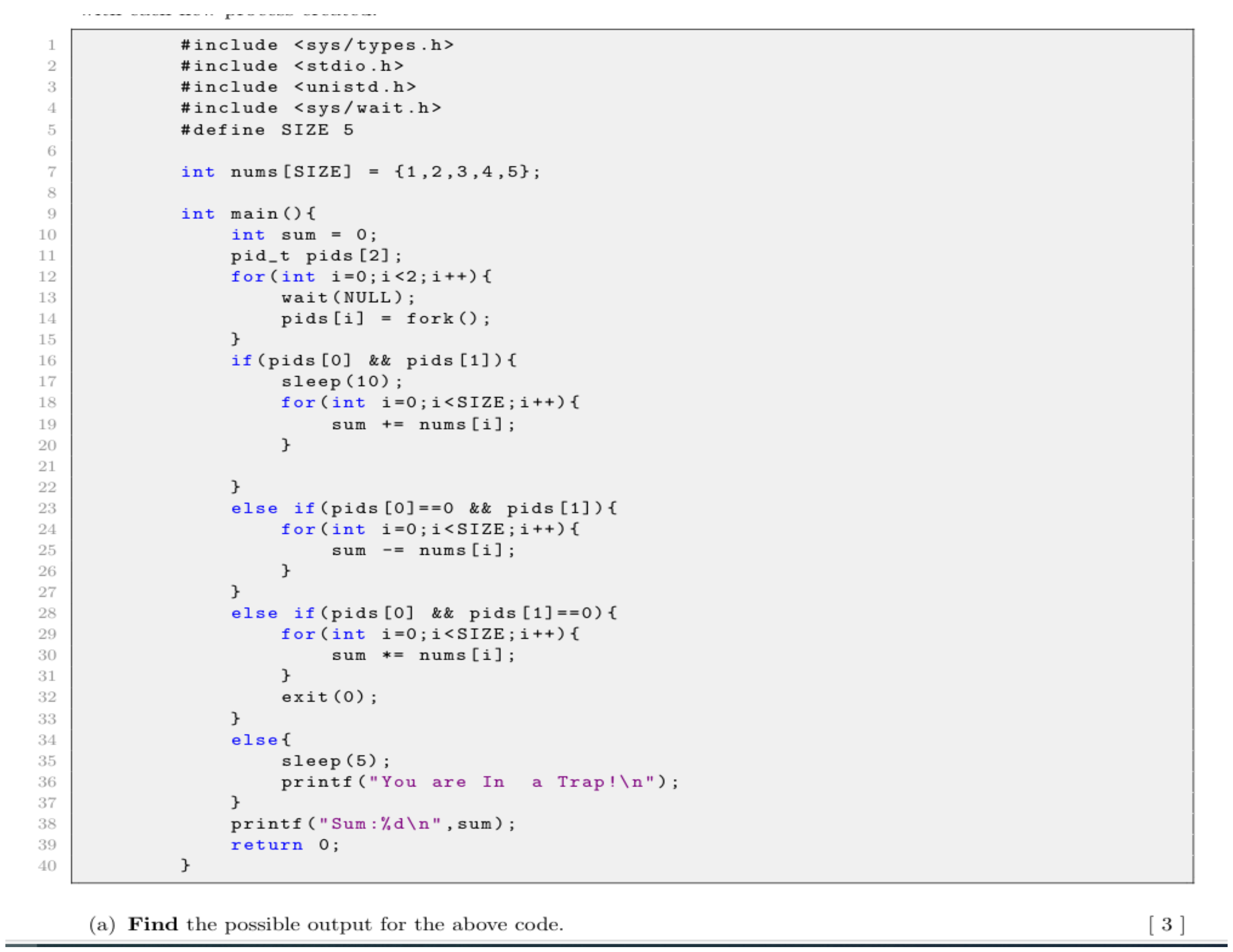
i. **Draw** a process tree for the above code.

ii. **Write** the possible output for the above code.

fork 1
fork 2
P
fork3
A, B, C
D, E, F



10)



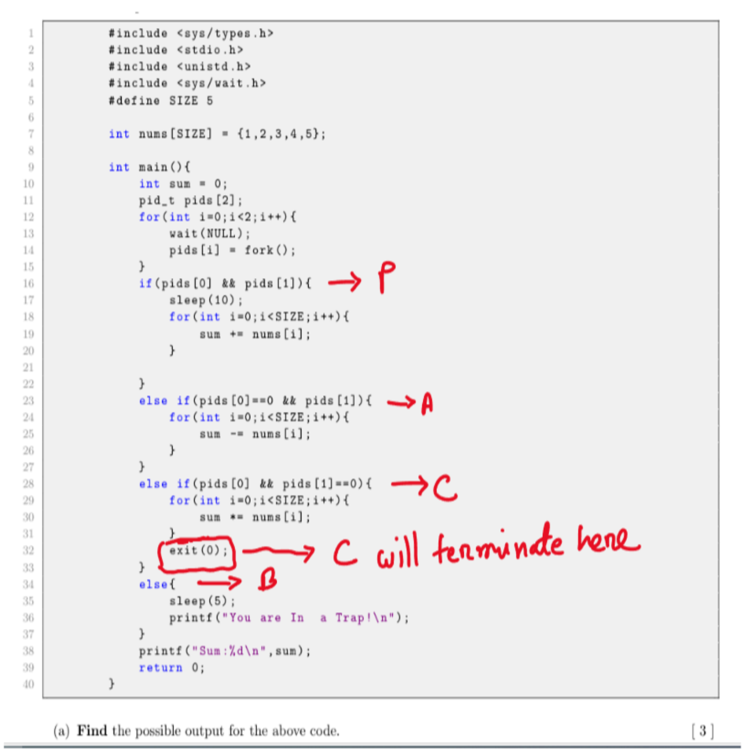
```c
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
#define SIZE 5

int nums[SIZE] = {1,2,3,4,5};

int main(){
    int sum = 0;
    pid_t pids[2];
    for(int i=0;i<2;i++){
        wait(NULL);
        pids[i] = fork();
    }
    if(pids[0] && pids[1]){
        sleep(10);
        for(int i=0;i<SIZE;i++){
            sum += nums[i];
        }

    }
    else if(pids[0]==0 && pids[1]){
        for(int i=0;i<SIZE;i++){
            sum -= nums[i];
        }
    }
    else if(pids[0] && pids[1]==0){
        for(int i=0;i<SIZE;i++){
            sum *= nums[i];
        }
        exit(0);
    }
    else{
        sleep(5);
        printf("You are In  a Trap!\n");
    }
    printf("Sum:%d\n",sum);
    return 0;
}
```

(a) **Find** the possible output for the above code. [3]



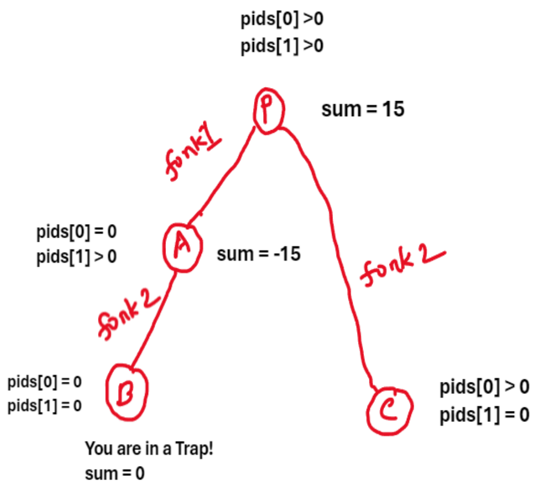
```c
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
#define SIZE 5

int nums[SIZE] = {1,2,3,4,5};

int main(){
    int sum = 0;
    pid_t pids[2];
    for(int i=0;i<2;i++){
        wait(NULL);
        pids[i] = fork();
    }
    if(pids[0] && pids[1]){            → P
        sleep(10);
        for(int i=0;i<SIZE;i++){
            sum += nums[i];
        }

    }
    else if(pids[0]==0 && pids[1]){    → A
        for(int i=0;i<SIZE;i++){
            sum -= nums[i];
        }
    }
    else if(pids[0] && pids[1]==0){    → C
        for(int i=0;i<SIZE;i++){
            sum *= nums[i];
        }
        exit(0);                       → C will terminate here
    }
    else{                              → B
        sleep(5);
        printf("You are In  a Trap!\n");
    }
    printf("Sum:%d\n",sum);
    return 0;
}
```

(a) **Find** the possible output for the above code. [3]



Possible Output:

```
Sum: -15
You are In a Trap!
Sum: 0
Sum: 15
```