WEEK-END ASSIGNMENT-S01
Processes in UNIX
Operating Systems Workshop (CSE 3541)

Problem Statement:

Experiment with programs, processes, memory allocation and manipulation for processes in UNIX.

Assignment Objectives:

Students will be able to differentiate programs and processes, also able to learn how to create processes and able to explore the implication of process inheritance.

Instruction to Students (If any):

Students are required to write his/her own program/output by avoiding any kind of copy from any sources. Additionally, They must be able to realise the outcome of that question in relevant to systems programming. You may use additional pages on requirement.

Programming/ Output Based Questions:

1. Construct the process tree diagram and also find the number of processes along with output of the following code snippet.

```
int main(void) {
    fork();
    fork();
    fork();
    printf("ITER\n");
    printf("ITER\n");
    return 0;
}
/* Any formula can be
   devised for number
    of processes
   created here?
If so, state.*/
```
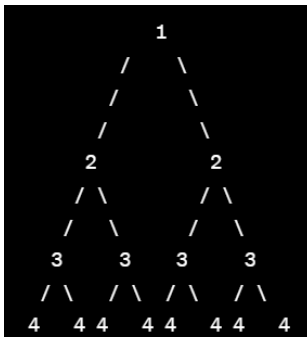
Process tree, # of processes and output

```
            1
           / \
          /   \
         2     2
        / \   / \
       3   3 3   3
```
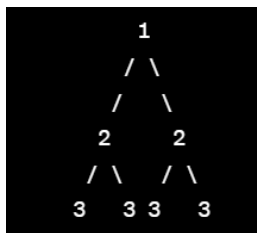
2. Construct the process tree diagram and also find the number of processes along with output of the following code snippet.

```
int main(void){
   printf("hello\n");
   fork();
   printf("hello\n");
   fork();
   printf("hello\n");
   fork();
   printf("hello\n");
   return 0;
}
/* Any formula for
    nuber of outputs?
If so, state.*/
```

Process tree, # of processes and output

```
            1
           / \
          /   \
         /     \
        2       2
       / \     / \
      /   \   /   \
     3    3  3     3
    / \  / \ / \  / \
   4  4 4  4 4 4 4   4
```
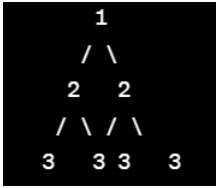
3. Run the following code on your machine and write the output. Suggest a way to avoid the mismatch of machine output w.r.t. dry run output.

```c
#include<stdio.h>
#include<unistd.h>
int main(void)
{
    printf("A");
    fork();
    printf("P\n");
    return 0;
}
```
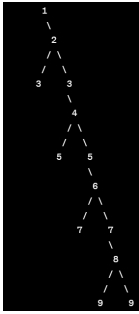
Output, Reason and Suggestion

4. Draw the process tree and write the output of the following code snippet.

```c
int main()
{
    fork() && fork();
    printf("Able to\n");
    return 0;
}
```
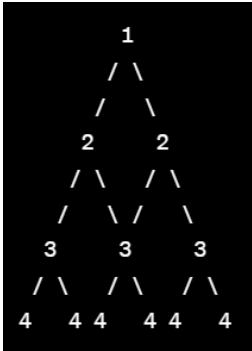
Process tree and output

```
       1
      / \
     2   2
    / \ / \
   3  3 3  3
```

5. Draw the process tree and write the output of the following code snippet.

```c
int main()
{
    fork();
    fork() && fork();
    fork();
    printf("Got!!!\n");
    return 0;
}
```

Process tree and output

```
   1
    \
     2
    / \
   3   3
    \
     4
    / \
   5   5
    \
     6
    / \
   7   7
    \
     8
    / \
   9   9
```

6. Draw the process tree and write the output of the following code snippet.

```c
int main()
{
    fork();
    fork() + fork();
    fork();
    printf("doing!\n");
    return 0;
}
```
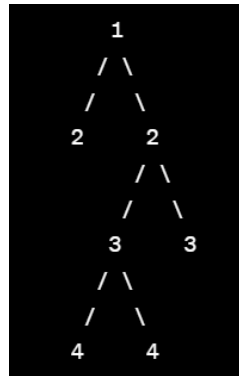
Process tree and output

```
           1
          / \
         /   \
        2     2
       / \   / \
      /   \ /   \
     3    3 3    3
    / \  / \   / \
   4  4 4  4 4   4
```

7. Draw the process tree and write the output of the following code snippet.

```
int main(){
    fork();
    fork() || fork();
    fork();
    printf("Really!!!\n");
    return 0;
}
```
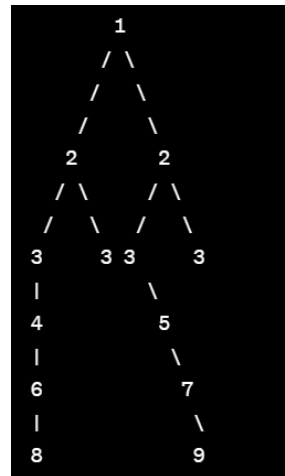
Process tree   **Output**

```
        1
       / \
      /   \
     2     2
          / \
         /   \
        3     3
       / \
      /   \
     4     4
```

Remark

8. Draw the process tree and write the output of the following code snippet.

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

Process tree   **Output**

```
         1
        / \
       /   \
      /     \
     2       2
    / \     / \
   /   \   /   \
  3    3 3      3
  |        \
  4         5
  |          \
  6           7
  |            \
  8             9
```

Remark
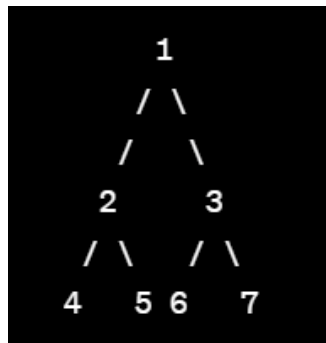
9. Draw the process tree and write the output of the following code snippet.

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

Process tree   **Output**

```
        1
       / \
      /   \
     2     3
    / \   / \
   4   5 6   7
```
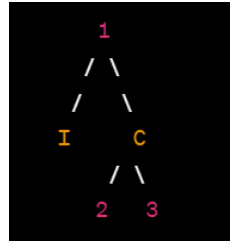
Remark

10. Construct the process tree diagram and also find the number of processes along with output of the following code snippet.

```c
int main(){
 int pid,pid2;
 pid=fork();
 if(pid){
   pid2=fork();
   printf("I\n");
 }
 else{
     printf("C\n");
     pid2=fork();
 }
 return 0;
}
```
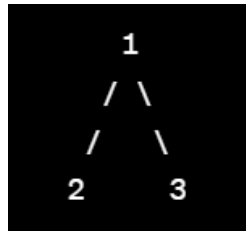
Process tree, # of processes and output		

11. Construct the process tree diagram and also find the number of processes along with output of the following code snippet.

```c
int main(void){
 pid_t childpid;
 int i, n=3;
 for(i=1;i<n;i++){
  childpid=fork();
  if(childpid==-1)
     break;
 }
 printf("i:%d\n",i);
 return 0;
}
```

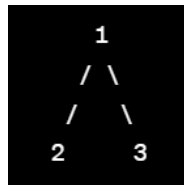| Process tree, # of processes and output		

12. Draw the process tree because of the following code snippet and state number of times **x=0** as well as **x≠0** will be displayed.

```c
pid_t
add(pid_t a, pid_t b){
   return a+b;
}
int main(void){
 pid_t x=10;
 printf("%d\n",x);
 x=add(fork(),fork());
 printf("%d\n",x);
 return 0;}
```

Process tree	# of x=0	# of x≠0

13. Determine the total number of displayed for the given code snippet.

```c
int main(void){
 int x[]={10,20,fork(),fork()+fork()};
 int len=sizeof(x)/sizeof(int);
 for(int i=0;i<len;i++)
   fprintf(stderr," %d ",x[i]);
 printf("\n");
 return 0;
}
```

# of display	# of Processes

14. Determine the number of process(s) will be created when the below program becomes process and also write the output.

```c
void show(){
  if(fork()==0)
    printf("1\n");
  if(fork()==0)
    printf("2\n");
  if(fork()==0)
    printf("3\n");
}
int main(void){
  show();
  return 0;
}
```

# of processes	Output

15. Draw the process tree of the following code snippet. Also give a count of processes and the output of the following code. Can the code segment generate fan of processes.

```c
int main(void){
  if(fork()==0)
    printf("1\n");
  else if(fork()==0)
    printf("2\n");
  else if(fork()==0)
    printf("3\n");
  else if(fork()==0)
    printf("4\n");
  else
    printf("5\n");
  return 0;
}
```

Process tree, # of processes and output

```
        1
       / \
      2   5
     / \
    3   5
   / \
  4   5
 / \
5   5
```

16. Find the output of the code segment showing the corresponding process tree.

```c
int main(){
  pid_t p1,p2;
  p2=0;
  p1=fork();
  if (p1 == 0)
    p2 = fork();
  if (p2 > 0)
    fork();
  printf("done\n");
  return 0;
}
```

Process tree   **Output**

```
     1
    / \
   2   3
      / \
     4   5
          \
           6
```

17. Find the number of direct children to the main process, the total number of processes and the output.

```c
int main(){pid_t c1=1,c2=1;
  c1=fork();
  if(c1!=0)
    c2=fork();
  if(c2==0){
    fork();printf("1\n");
  }
  return 0;}
```

# of direct children to main process	Total processes	Output

18. Find the output of the code segment.

```c
int main(){
  struct stud s1={1,20};
  pid_t pid=fork();
  if(pid==0){
     struct stud s1={2,30};
     printf("%d %d\n",s1.r,s1.m);
     return 0;
  }
  else{
     sleep(10);
     printf("%d %d\n",s1.r,s1.m);
     return 0;
  }
}
```
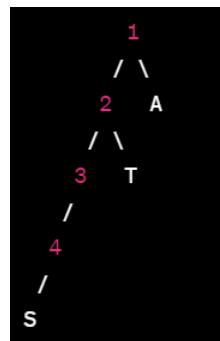
```c
struct stud{
  int r;
  int m;
};
```

Output

19. Find the output of the code segment showing the corresponding process tree.

```c
int main(){
  if(fork()){
     if(!fork()){
        fork();
        printf("S ");
     }
     else{
        printf("T ");
     }
  }
  else{
        printf("D ");
  }
  printf("A ");
  return 0;
}
```
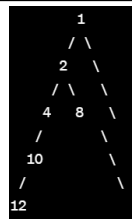
Process tree   **Output**

```
        1
       / \
      2   A
     / \
    3   T
   /
  4
 /
S
```

20. Calculate the number of processes the following code snippet will generate.

```c
int main(){int i;
  for(i=0;i<12;i++){
     if(i%3==0){
           fork();
     }
  }
  return 0;
}
```

Process tree   **Output**

```
       1
      / \
     2   \
    / \   \
   4   8   \
  /       \
 10        \
 /          \
12
```

21. State the possible values of x for the given code snippet;

Process tree   **Output**

```c
int x;
int a[2]={10,20};
x=5+a[fork() || fork()];
printf("%d ",x);
```

```
    1
   / \
  /   \
 2     3
```

22. Suppose four user-defined exit handlers X, Y, P, and Q are installed in the order X then Y then P then Q using atexit() function in a C program. Exit hadler X is designed to display 1, Y is designed to display 2, P is designed to display 3, and Q to display 4. State the order of their display, when the program is going to terminate after calling return 0/exit(0).

(A) 4, 3, 2 1   (C) 1, 2, 4, 3

(B) 1,2,3,4   (D) none

Choice

23. You know that the **ps** utility in UNIX reports a snapshot of the current processes. Determine the state code of the given program, that became a process.

```
int main(void){
 fprintf(stderr,"PID=%ld\n",(long)getpid());
 while(1);
 return 0;
}
```

(A) R   (C) T

(B) S   (D) Z

Choice

24. Find the process state code of the given program, that became a process using the Unix utility **ps**. As you know **ps** displays information about a selection of the active processes.

```
int main(void){
 fprintf(stderr,"PID=%ld\n",(long)getpid());
 while(1)
      sleep(1);
 return 0;
}
```

(A) R   (C) T

(B) S   (D) Z

Choice

25. Develop a C code to create the following process tree. Display the process ID, parent ID and return value of fork() for each process.
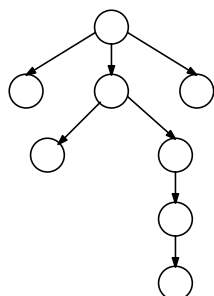
OBSERVARIONS:

- Use ps utility to verify the is-a-parent relationship?
- Are ypu gettig any orphan process case?
- Are you getting any ZOMBIE case?

Figure 1: Process tree

Code here

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
int main() {
    pid_t pid1, pid2, pid3, pid4, pid5, pid6;
    printf("Initial process:\n");
    printf("PID: %d, PPID: %d, fork() return value: %d\n", getpid(), getppid(), 0);
    pid1 = fork();
    if (pid1 < 0) {
        perror("fork");
        exit(1);
    } else if (pid1 == 0) {
        printf("Process 1:\n");
        printf("PID: %d, PPID: %d, fork() return value: %d\n", getpid(), getppid(), pid1);
        pid4 = fork();
        if (pid4 < 0) {
            perror("fork");
            exit(1);
        } else if (pid4 == 0) {
            printf("Process 4:\n");
            printf("PID: %d, PPID: %d, fork() return value: %d\n", getpid(), getppid(), pid4);
            exit(0);
        }
        pid5 = fork();
        if (pid5 < 0) {
            perror("fork");
            exit(1);
        } else if (pid5 == 0) {
            printf("Process 5:\n");
            printf("PID: %d, PPID: %d, fork() return value: %d\n", getpid(), getppid(), pid5);
            exit(0);
        }
        waitpid(pid4, NULL, 0);
        waitpid(pid5, NULL, 0);
        exit(0);
    }
    pid2 = fork();
    if (pid2 < 0) {
        perror("fork");
        exit(1);
    } else if (pid2 == 0) {
        printf("Process 2:\n");
        printf("PID: %d, PPID: %d, fork() return value: %d\n", getpid(), getppid(), pid2);
        exit(0);
    }
    pid3 = fork();
    if (pid3 < 0) {
        perror("fork");
        exit(1);
    } else if (pid3 == 0) {
        printf("Process 3:\n");
        printf("PID: %d, PPID: %d, fork() return value: %d\n", getpid(), getppid(), pid3);
        pid6 = fork();
        if (pid6 < 0) {
            perror("fork");
            exit(1);
        } else if (pid6 == 0) {
            printf("Process 6:\n");
            printf("PID: %d, PPID: %d, fork() return value: %d\n", getpid(), getppid(), pid6);
            exit(0);
        }
        waitpid(pid6, NULL, 0);
        exit(0);
    }
    waitpid(pid1, NULL, 0);
    waitpid(pid2, NULL, 0);
    waitpid(pid3, NULL, 0);
    printf("All child processes have terminated.\n");
    return 0;
}
```

26. Create two different user-defined functions to generate the following process hierarchy shown in **Figure-(a)** and **Figure-(b)**. Finally all the processes display their process ID and parent ID.
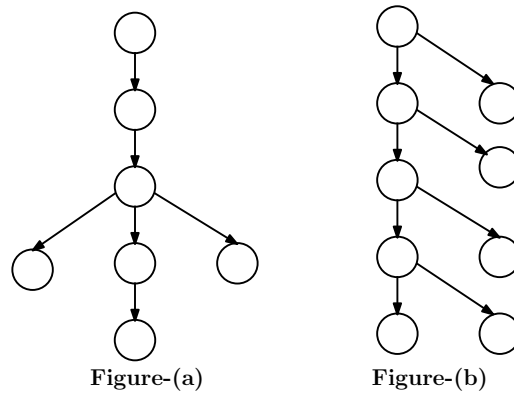

Figure-(a)   Figure-(b)

Figure 2: Process tree

Code here

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

void figureA() {
    pid_t child1, child2, child3;

    if((child1 = fork()) == 0) {
        printf("Process ID: %d, Parent ID: %d\n", getpid(), getppid());
        exit(0);
    }

    if((child2 = fork()) == 0) {
        printf("Process ID: %d, Parent ID: %d\n", getpid(), getppid());
        exit(0);
    }

    if((child3 = fork()) == 0) {
        printf("Process ID: %d, Parent ID: %d\n", getpid(), getppid());
        exit(0);
    }

    printf("Parent Process ID: %d\n", getpid());
}
```

Code here

```c
void figureB() {
    pid_t child1;

    if((child1 = fork()) == 0) {

        pid_t grandChild;

        if ((grandChild = fork()) == 0){
            printf("Process ID: %d, Parent ID: %d\n", getpid(), getppid());
            exit(0);
        }

        if ((grandChild = fork()) == 0){
            printf("Process ID: %d, Parent ID: %d\n", getpid(), getppid());
            exit(0);
        }

        printf("Process ID: %d, Parent ID: %d\n", getpid(), getppid());
        exit(0);
    }

    if((child1 = fork()) == 0) {
        printf("Process ID:% d, ParentID:% d \n ",getpid (),getppid ());
        exit ( 0 );
    }

    printf ("Parent ProcessID:% d \n ",getpid ());
}

int main() {

    printf("\nFigure-(a)\n");

    figureA();

    sleep(5); // To separate the outputs of two figures

    printf("\nFigure-(b)\n");

    figureB();

    return 0;
}
```

27. What output will be at Line X and Line Y?

```c
#define SIZE 5
int nums[SIZE] = { 0,1,2,3,4 } ;
int main(){
 int i;
 pid t pid;
 pid = fork();
 if(pid == 0){
    for (i = 0; i < SIZE; i++) {
        nums[i] *= nums[i] *-i;
        printf("CHILD:%d ",nums[i]);  /* LINE X */
    }
 }
 else if (pid > 0) {
    wait(NULL);
    for (i = 0; i < SIZE; i++)
        printf("PARENT: %d ",nums[i]);  /* LINE Y */
    }
  return 0;
}
```

Output

28. The Fibonacci sequence is the series of numbers 0, 1, 1, 2, 3, 5, 8,

Write a C program using the fork() system call that generates the Fibonacci sequence in the child process. The number of the sequence will be provided in the command line. For example, if 5 is provided, the first five numbers in the Fibonacci sequence will be output by the child.

Code here

29. Write a **MulThree.c** program to multiply three numbers and display the output. Now write another C program **PracticeExecl.c**, which will fork a child process and the child process will execute the file **MulThree.c** and generate the output. The parent process will wait till the termination of the child and the parent process will print the process ID and exit status of the child.

Code here

30. You know the usages of the command **grep**. Implement the working of **grep -n pattern filename** in a child process forked from the parent process using **execl** system call. The parent process will wait till the termination of the child and the parent process will print the process ID and exit status of the child.

Code here

31. Implement the above question using **execv**, **execlp**, **execvp**, **execle**, **execve** system calls.

Code here for execv

Code here for `execlp`

Code here for `execvp`

Code here for `execve`

Code here for `execle`