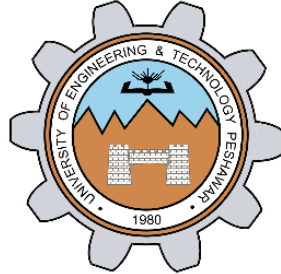


LAB NO 03
PROCESSES IN UNIX



Fall 2024
CSE-302L Systems Programming Lab

Submitted by:

Name: **Hassan Zaib Jadoon**

Reg no: **22PWCSE2144**

Class Section: **A**

Signature: _____

Submitted to:

Engr. Abdullah Hamid

December 29, 2024

Department of Computer Systems Engineering
University of Engineering and Technology, Peshawar

LAB NO 03 PROCESSES IN UNIX

Task 1

Create process chain as shown in figure 3.1(b) and fill the figure 3.1 (b) with actual IDs. The program shall take a single command-line argument that specifies the number of processes to be created. Before exiting, each process shall output its i value (loop variable), its process ID (using getpid()), its parent process ID (getppid()) and the process ID of its child (return value of fork). The parent does not execute wait.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int main(int argc, char *argv[]) {
    int n = atoi(argv[1]);
    pid_t pid;
    for (int i = 0; i < n; i++) {
        pid = fork();
        if (pid < 0) {
            perror("fork");
            return 1;
        }
        if (pid == 0) {
            printf("Process %d: PID = %d, PPID = %d\n", i, getpid(), getppid());
        } else {
            exit(0);
        }
    }
    return 0;
}
```

Output:

```
hassan@hassan-HP-ProBook-4740s:~/Desktop/SP Lab/SP Lab 03$ gcc task1.c -o task1.o
hassan@hassan-HP-ProBook-4740s:~/Desktop/SP Lab/SP Lab 03$ ./task1.o 6
Process 0: PID = 3971, PPID = 3970
Process 1: PID = 3972, PPID = 3971
Process 2: PID = 3973, PPID = 3972
Process 3: PID = 3974, PPID = 1188
Process 4: PID = 3975, PPID = 3974
Process 5: PID = 3976, PPID = 3975
```

Task 2

Create process fan as shown in figure 3.1 (a) and fill the figure 3.1 (a) with actual IDs.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
int main(int argc, char *argv[]) {
    int n = atoi(argv[1]);
    pid_t pid;
    for (int i = 0; i < n; i++) {
        pid = fork();
        if (pid < 0) {
            perror("Fork failed");
            exit(1);
        } else if (pid == 0) {
            printf("Child %d: PID = %d, Parent PID = %d\n", i + 1, getpid(), getppid());
            exit(0);
        }
    }
    for (int i = 0; i < n; i++) {
        wait(NULL);
    }
    return 0;
}
```

Output

```
hassan@hassan-HP-ProBook-4740s:~/Desktop/SP Lab/SP Lab 03$ gcc task2.c -o task2.o
hassan@hassan-HP-ProBook-4740s:~/Desktop/SP Lab/SP Lab 03$ ./task2.o
Segmentation fault (core dumped)
hassan@hassan-HP-ProBook-4740s:~/Desktop/SP Lab/SP Lab 03$ ./task2.o 3
Child 2: PID = 4476, Parent PID = 4474
Child 1: PID = 4475, Parent PID = 4474
Child 3: PID = 4477, Parent PID = 4474
```

Task 3

Create process tree as shown in figure 3.2 and fill figure 3.2 with actual IDs.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
void create_children(int level, int num_children) {
    for (int i = 0; i < num_children; i++) {
        pid_t pid = fork();
        if (pid < 0) {
            perror("Fork failed");
            exit(1);
        } else if (pid == 0) {
            printf("Level %d Child %d: PID = %d, Parent PID = %d\n", level, i + 1, getpid(), getppid());
            if (level == 2 && (i == 0 || i == 1)) {
                create_children(3, 2);
            } else if (level == 3 && (i == 0 || i == 1)) {
                create_children(4, 2);
            }
            exit(0);
        }
    }
}
for (int i = 0; i < num_children; i++) {
    wait(NULL);
}
int main() {
    printf("Root process: PID = %d\n", getpid());
    create_children(2, 3);
    return 0;
}
```

Output:

```
hassan@hassan-HP-ProBook-4740s:~/Desktop/SP Lab/SP Lab 03$ ./task4.o 4
Process 1: PID = 4326, Parent PID = 3937, Child PID = 4327
Process 2: PID = 4327, Parent PID = 4326, Child PID = 4328
Process 3: PID = 4328, Parent PID = 4327, Child PID = 4329
Process 4: PID = 4329, Parent PID = 1188, Child PID = 4330
hassan@hassan-HP-ProBook-4740s:~/Desktop/SP Lab/SP Lab 03$
```

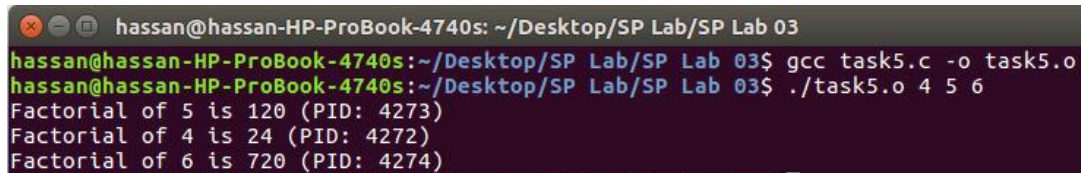
Task 5

Write a program that takes N number of integers as argument and displays the factorials of N integers (print 1 only if integers are not less than zero, 0 or 1). Create separate child process for each integer. Make sure no child is orphan/zombie.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
long factorial(int n) {
    if (n == 0 || n == 1) return 1;
    long result = 1;
    for (int i = 2; i <= n; i++) {
        result *= i;
    } return result;
}
int main(int argc, char *argv[]) {
    for (int i = 1; i < argc; i++) {
        int num = atoi(argv[i]);
        pid_t pid = fork();
        if (pid < 0) {
            perror("Fork failed");
            exit(1);
        } else if (pid == 0) {
            long fact = factorial(num);
            printf("Factorial of %d is %ld (PID: %d)\n", num, fact, getpid());
            exit(0);
        }
    }
    for (int i = 1; i < argc; i++) {
        wait(NULL);
    } return 0;
}
```

Output:



```
hassan@hassan-HP-ProBook-4740s: ~/Desktop/SP Lab/SP Lab 03
hassan@hassan-HP-ProBook-4740s:~/Desktop/SP Lab/SP Lab 03$ gcc task5.c -o task5.o
hassan@hassan-HP-ProBook-4740s:~/Desktop/SP Lab/SP Lab 03$ ./task5.o 4 5 6
Factorial of 5 is 120 (PID: 4273)
Factorial of 4 is 24 (PID: 4272)
Factorial of 6 is 720 (PID: 4274)
```

Task 6

Write a program that takes N number of integers as argument and displays the factorials of N integers (print 1 only if integers are not less than zero, 0 or 1). Create separate child process for each integer. Make sure no child is orphan/zombie.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
void fill_array(int *arr) {
    arr[0] = 42; arr[1] = 33; arr[2] = 99; arr[3] = 16; arr[4] = 86; arr[5] = 12; arr[6] = 75;
    arr[7] = 4; arr[8] = 51; arr[9] = 63; arr[10] = 86; arr[11] = 94; arr[12] = 21; arr[13] = 7; arr[14] = 2; arr[15] = 87;
}
int sum_array(int *arr, int start, int end) {
    int sum = 0;
    for (int i = start; i < end; i++) {
        sum += arr[i];
    }
    return sum;
}
int main() {
    int arr[15];
    fill_array(arr);
    int total_sum = 0;
    int num_children = 10;
    for (int i = 0; i < num_children; i++) {
        pid_t pid = fork();
        if (pid == 0) {
            int partial_sum = sum_array(arr, i * (15 / num_children), (i + 1) * (15 / num_children));
            printf("Child %d: Partial sum = %d\n", i, partial_sum);
            if (partial_sum > 50) { exit(partial_sum + 5); }
        } else {
            exit(partial_sum);
        }
    }
    for (int i = 0; i < num_children; i++) {
        int status; wait(&status);
        total_sum += WEXITSTATUS(status);
    }
    printf("Total sum: %d\n", total_sum);
    return 0;
}
```

Output:

```
hassan@hassan-HP-ProBook-4740s: ~/Desktop/SP Lab/SP Lab 03
hassan@hassan-HP-ProBook-4740s:~/Desktop/SP Lab/SP Lab 03$ gcc task6.c -o task6.o
hassan@hassan-HP-ProBook-4740s:~/Desktop/SP Lab/SP Lab 03$ ./task6.o
Child 0: Partial sum = 42
Child 1: Partial sum = 33
Child 3: Partial sum = 16
Child 4: Partial sum = 86
Child 5: Partial sum = 12
Child 7: Partial sum = 4
Child 2: Partial sum = 99
Child 8: Partial sum = 51
Child 6: Partial sum = 75
Child 9: Partial sum = 63
Total sum: 506
*** stack smashing detected ***: ./task6.o terminated
Aborted (core dumped)
hassan@hassan-HP-ProBook-4740s:~/Desktop/SP Lab/SP Lab 03$
```

CSE 302L: SYSTEMS PROGRAMMING LAB

LAB ASSESSMENT RUBRICS

Criteria & Point Assigned	Outstanding 2	Acceptable 1.5	Considerable 1	Below Expectations 0.5	Score
Attendance and Attentiveness in Lab PLO08	Attended in proper Time and attentive in Lab	Attended in proper Time but not attentive in Lab	Attended late but attentive in Lab	Attended late not attentive in Lab	
Capability of writing Program/ Algorithm/Drawing Flow Chart PLO1, PLO2, PLO3, PLO5,	Right attempt/ no errors and well formatted	Right attempt/ no errors but not well formatted	Right attempt/ minor errors and not well formatted	Wrong attempt	
Result or Output/ Completion of target in Lab PLO9,	100% target has been completed and well formatted.	75% target has been completed and well formatted.	50% target has been completed but not well formatted.	None of the outputs are correct	
Overall, Knowledge PLO10,	Demonstrates excellent knowledge of lab	Demonstrates good knowledge of lab	Has partial idea about the Lab and procedure followed	Has poor idea about the Lab and procedure followed	
Attention to Lab Report PLO4,	Submission of Lab Report in Proper Time i.e., in next day of lab., with proper documentation.	Submission of Lab Report in proper time but not with proper documentation.	Late Submission with proper documentation.	Late Submission Very poor documentation	

Instructor:

Name: _____

Signature: _____