**Exp5**

1. **Write a C program to illustrate that performing ’n’ consecutive fork() system calls generates a total of 2n − 1 child processes. The program should prompt the user to input the value of ’n’.”**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

int main() {

int n;

printf("Enter the value of 'n': ");

scanf("%d", &n);

if (n <= 0) {

printf("Invalid input. 'n' should be a positive integer.\n");

return 1;

}

int i;

pid\_t pid;

for (i = 0; i < n; i++) {

pid = fork();

if (pid < 0) {

printf("Fork failed.\n");

return 1;

} else if (pid == 0) {

// Child process

printf("Child process created with PID: %d\n", getpid());

break; // Exit the loop in child processes

} else {

// Parent process

printf("Parent process with PID: %d created child with PID: %d\n", getpid(), pid);

}

}

// Wait for all child processes to terminate

while (wait(NULL) > 0);

return 0;

}

1. **Write a C program utilizing the fork() system call to generate the following process hierarchy: P1 → P2 → P3. The program should display the Process ID (PID) and Parent Process IDs (PPID) for each process created.**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

int main() {

printf("P1: PID=%d, PPID=%d\n", getpid(), getppid());

pid\_t pid1 = fork(); // Create P2

if (pid1 < 0) {

fprintf(stderr, "Fork failed\n");

return 1;

} else if (pid1 == 0) { // Child process (P2)

printf("P2: PID=%d, PPID=%d\n", getpid(), getppid());

pid\_t pid2 = fork(); // Create P3

if (pid2 < 0) {

fprintf(stderr, "Fork failed\n");

return 1;

} else if (pid2 == 0) { // Child process (P3)

printf("P3: PID=%d, PPID=%d\n", getpid(), getppid());

}

} else { // Parent process (P1)

wait(NULL); // Wait for child process (P2) to finish

}

return 0;

}

1. **Write a C program to generate a process hierarchy as follows: P1 P2 P4 P5 P3 The program should create the specified process structure using the appropriate sequence of ‘fork()‘ system calls. Print PID and PPID of each process.**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

int main() {

printf("P1: PID=%d, PPID=%d\n", getpid(), getppid());

pid\_t pid2 = fork(); // Create P2

if (pid2 < 0) {

fprintf(stderr, "Fork failed\n");

return 1;

} else if (pid2 == 0) { // Child process (P2)

printf("P2: PID=%d, PPID=%d\n", getpid(), getppid());

pid\_t pid4 = fork(); // Create P4

if (pid4 < 0) {

fprintf(stderr, "Fork failed\n");

return 1;

} else if (pid4 == 0) { // Child process (P4)

printf("P4: PID=%d, PPID=%d\n", getpid(), getppid());

} else { // Parent process (P2)

wait(NULL); // Wait for child process (P4) to finish

pid\_t pid5 = fork(); // Create P5

if (pid5 < 0) {

fprintf(stderr, "Fork failed\n");

return 1;

} else if (pid5 == 0) { // Child process (P5)

printf("P5: PID=%d, PPID=%d\n", getpid(), getppid());

} else { // Parent process (P2)

wait(NULL); // Wait for child process (P5) to finish

}

}

} else { // Parent process (P1)

wait(NULL); // Wait for child process (P2) to finish

pid\_t pid3 = fork(); // Create P3

if (pid3 < 0) {

fprintf(stderr, "Fork failed\n");

return 1;

} else if (pid3 == 0) { // Child process (P3)

printf("P3: PID=%d, PPID=%d\n", getpid(), getppid());

} else { // Parent process (P1)

wait(NULL); // Wait for child process (P3) to finish

}

}

return 0;

}