📘 C Programming Basics

Lab Session: System Call Timing and Process Synchronization

Course: Operating Systems
Date: 26.05.2025

Program 1: Sleep and Exit

```
#include <stdio.h>
#include <unistd.h> // for sleep()
#include <stdlib.h> // for exit()

int main(){
   printf("Program started.\n");

   printf("Sleeping for 3 seconds...\n");
   sleep(3); // pause for 3 seconds

   printf("Exiting the program.\n");
   exit(0); // clean exit

   return 0;
}
```

Fedora Output:

```
[2021ict108@fedora ~]$ vi Cpro11.c
[2021ict108@fedora ~]$ gcc Cpro11.c -o Cpro11
[2021ict108@fedora ~]$ ./Cpro11
Program started.
```

Sleeping for 3 seconds...

Exiting the program.

Explanation:

- This program demonstrates how to use the sleep() function to pause execution for a number of seconds.
- After sleeping for 3 seconds, the program terminates using exit(0), indicating a normal, successful termination.
- Useful for simulating delays or waiting behavior in real-time processes.
  Explanation:
- Demonstrates use of sleep() and exit().
- Pauses for 3 seconds before exiting cleanly.

Program 2: Wait for Child Exit

```
#include <stdio.h>

#include <unistd.h> // for sleep()

#include <stdlib.h> // for exit()

#include <sys/wait.h> // for wait()


int main(){

   pid_t pid;


   printf("Parent process started. PID: %d\n", getpid());


   pid = fork();


   if(pid < 0){

      perror("fork failed");
```

```
        exit(1);
    }

    if (pid == 0){
        // Child process
        printf("I am Child. My parent ID: %d\n", getppid());
        printf("Child process. PID: %d, sleeping for 2 seconds...", getpid());
        sleep(2);
        printf("Child process exiting.\n");
        exit(0);
    } else {
        // Parent process
        int status;
        printf("Parent waiting for child to finish...\n");
        wait(&status);

        if(WIFEXITED(status)){
            printf("Child exited with status: %d\n", WEXITSTATUS(status));
        } else {
            printf("Child did not exit normally.\n");
        }
        printf("Parent process ending.\n");
    }

    return 0;
}
```

Fedora Output:

[2021ict108@fedora ~]$ vi Cpro12.c

[2021ict108@fedora ~]$ gcc Cpro12.c -o Cpro12

[2021ict108@fedora ~]$ ./Cpro12

Parent process started. PID: 12340

Parent waiting for child to finish...

I am Child. My parent ID: 12340

Child process. PID: 12341, sleeping for 2 seconds...Child process exiting.

Child exited with status: 0

Parent process ending.

Explanation:

- This program creates a child process using fork().
- The child process sleeps for 2 seconds and then exits.
- The parent uses wait() to pause until the child process finishes.
- WIFEXITED(status) checks if the child terminated normally.
- WEXITSTATUS(status) retrieves the child's exit code (which is 0 here).
- This structure is commonly used to ensure that parent processes don't terminate before their children.

Program 3: Two Children with Different Sleep Durations

```
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>


int main() {
```

```
pid_t pid1, pid2;

printf("Parent process started. PID: %d\n", getpid());

pid1 = fork();
if (pid1 < 0) {
    perror("fork failed");
    exit(1);
}

if (pid1 == 0) {
    printf("First child: PID %d, sleeping for 1 second...\n", getpid());
    sleep(1);
    printf("First child: Finished sleeping.\n");
    exit(0);
}

pid2 = fork();
if (pid2 < 0) {
    perror("fork failed");
    exit(1);
}

if (pid2 == 0) {
    printf("Second child: PID %d, sleeping for 3 seconds...\n", getpid());
    sleep(3);
```

```
        printf("Second child: Finished sleeping.\n");
        exit(0);
    }

    int status;

    printf("Parent waiting for first child to finish...\n");
    waitpid(pid1, &status, 0);
    if (WIFEXITED(status)) {
        printf("First child exited with status: %d\n", WEXITSTATUS(status));
    }

    printf("Parent waiting for second child to finish...\n");
    waitpid(pid2, &status, 0);
    if (WIFEXITED(status)) {
        printf("Second child exited with status: %d\n", WEXITSTATUS(status));
    }

    printf("Parent: Both children have finished.\n");
    return 0;
}
```

Fedora Output:

```
[2021ict108@fedora ~]$ vi Cpro14.c
[2021ict108@fedora ~]$ gcc Cpro14.c -o Cpro14
[2021ict108@fedora ~]$ ./Cpro14
Parent process started. PID: 14513
```

Parent waiting for first child to finish...

First child: PID 14514, sleeping for 1 second...

Second child: PID 14515, sleeping for 3 seconds...

First child: Finished sleeping.

First child exited with status: 0

Parent waiting for second child to finish...

Second child: Finished sleeping.

Second child exited with status: 0

Parent: Both children have finished.

Explanation:

- Two child processes are created from the parent.
    - First child sleeps for 1 second.
    - Second child sleeps for 3 seconds.
- The parent uses waitpid() to wait for each child specifically by PID.
- The program showcases:
    - How to manage multiple children.
    - How to wait for them in a controlled sequence.
    - Handling and reporting individual child exit statuses.
- It clearly demonstrates process synchronization and how independent child processes can be monitored by the parent.
