22BCE0744

Operating System LAB DA2

1. **Aim:** To demonstrate a scenario where a parent process creates a child process and terminates before the child process finishes execution, showing how the child becomes an orphan and is adopted by the init process.

Pseudocode:

1. Start main function.
2. Call fork() to create a new process.
   - If fork() fails, print error and exit.
   - If in child process:
     a. Print child's PID and parent's PID.
     b. Flush output, sleep 2 seconds.
     c. Print child's PID and new parent's PID.
     d. Flush output.
   - If in parent process:
     a. Sleep 1 second.
     b. Print parent's PID and child's PID.
     c. Flush output, then exit.
3. End main function.

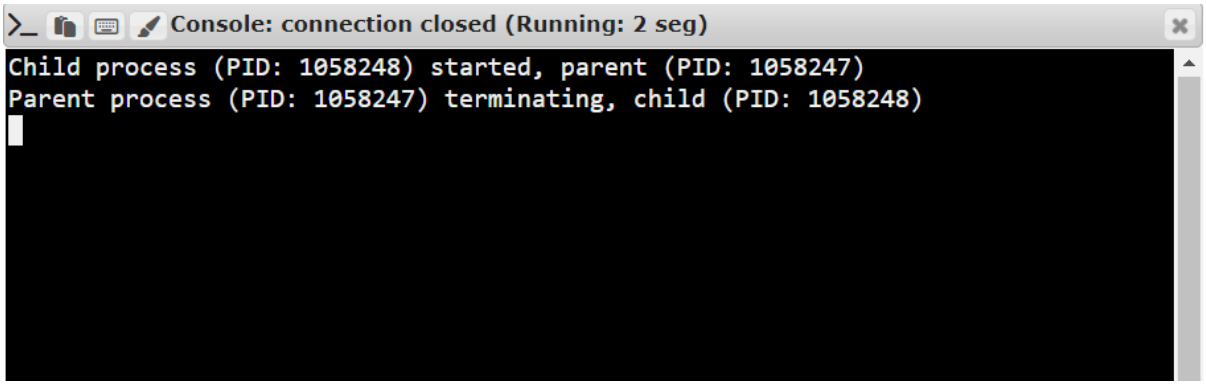
Code:

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

int main()
{
   pid_t pid = fork();
   if (pid < 0)
   {
      fprintf(stderr, "Fork failed\n");
      return 1;
   }
   else if (pid == 0)
   {
      printf("Child process (PID: %d) started, parent (PID: %d)\n", getpid(), getppid());
      fflush(stdout);
      sleep(2);
      printf("Child process (PID: %d) ending, new parent (PID: %d)\n", getpid(),
getppid());
```

```
        fflush(stdout);
    }
    else
    {
        sleep(1);
        printf("Parent process (PID: %d) terminating, child (PID: %d)\n", getpid(), pid);
        fflush(stdout);
        exit(0);
    }
}
```

Output:



2. **Aim:** Write a C program to simulate Round Robin Scheduling Algorithm

Pseudocode:

1. Define Process struct with attributes like id, burst_time, arrival_time, etc.
2. Define function to sort processes by arrival time.
3. Define Round Robin scheduler function:
   a. Initialize current time and completed processes count.
   b. Sort processes by arrival time.
   c. Add the first process to the queue.
   d. While not all processes are completed:
      - Dequeue the next process.
      - If remaining time <= time quantum:
        - Update current time, mark process as completed.
      - Else:
        - Reduce remaining time by time quantum, update current time.
      - Enqueue any arrived and uncompleted processes.
      - Re-enqueue the current process if not completed.
   e. Calculate turnaround and waiting times for each process.
4. In main function:
   a. Input number of processes and their burst/arrival times.
   b. Call Round Robin scheduler.
   c. Print process details (burst, arrival, completion, turnaround, waiting times).
5. End main function.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_PROCESSES 10
#define TIME_QUANTUM 2
struct Process
{
   int id;
   int burst_time;
   int arrival_time;
   int remaining_time;
   int completion_time;
   int turnaround_time;
   int waiting_time;
};
void sort_processes_by_arrival(struct Process processes[], int n)
{
   for (int i = 0; i < n - 1; i++)
   {
      for (int j = 0; j < n - i - 1; j++)
      {
         if (processes[j].arrival_time > processes[j + 1].arrival_time)
         {
            struct Process temp = processes[j];
            processes[j] = processes[j + 1];
            processes[j + 1] = temp;
         }
      }
   }
}
void round_robin_scheduler(struct Process processes[], int n)
{
   int current_time = 0;
   int completed = 0;
   int queue[MAX_PROCESSES], front = 0, rear = 0;
   sort_processes_by_arrival(processes, n);
   queue[rear++] = 0;
   while (completed < n)
   {
      int current_process = queue[front];
      front = (front + 1) % MAX_PROCESSES;
      if (processes[current_process].remaining_time <= TIME_QUANTUM)
      {
         current_time += processes[current_process].remaining_time;
```

```
            processes[current_process].remaining_time = 0;
            processes[current_process].completion_time = current_time;
            completed++;
        }
        else
        {
            current_time += TIME_QUANTUM;
            processes[current_process].remaining_time -= TIME_QUANTUM;
        }
        for (int i = 0; i < n; i++)
        {
            if (processes[i].arrival_time <= current_time && processes[i].remaining_time >
0 &&
                i != current_process && i != queue[(rear - 1 + MAX_PROCESSES) %
MAX_PROCESSES])
            {
                queue[rear] = i;
                rear = (rear + 1) % MAX_PROCESSES;
            }
        }

        if (processes[current_process].remaining_time > 0)
        {
            queue[rear] = current_process;
            rear = (rear + 1) % MAX_PROCESSES;
        }
    }

    for (int i = 0; i < n; i++)
    {
        processes[i].turnaround_time = processes[i].completion_time -
                                       processes[i].arrival_time;
        processes[i].waiting_time = processes[i].turnaround_time -
processes[i].burst_time;
    }
}
int main()
{
    int n;
    struct Process processes[MAX_PROCESSES];
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    printf("Enter burst time and arrival time for each process:\n");
    for (int i = 0; i < n; i++)
```

```
    {
        processes[i].id = i + 1;
        printf("P%d: ", i + 1);
        scanf("%d %d", &processes[i].burst_time, &processes[i].arrival_time);
        processes[i].remaining_time = processes[i].burst_time;
    }
    round_robin_scheduler(processes, n);
 printf("\nProcess\tBurst Time\tArrival Time\tCompletion
Time\tTurnaroundTime\tWaiting Time\n");
 for (int i = 0; i < n; i++) {
        printf("P%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n",
               processes[i].id, processes[i].burst_time, processes[i].arrival_time,
               processes[i].completion_time, processes[i].turnaround_time,
               processes[i].waiting_time);
 }
 return 0;
}
```

Output:

```
Console: connection closed (Running: 8 seg)
Enter the number of processes: 3
Enter burst time and arrival time for each process:
P1: 10
3
P2: 2
1
P3: 6
0

Process Burst Time      Arrival Time    Completion Time TurnaroundTime  Waiting
Time
P3      6               0               10              10              4
P2      2               1               4               3               1
P1      10              3               18              15              5
```