Slip 20

Program 1: Fork Process and Display Parent-Child Process ID

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

int main() {
    pid_t pid = fork();

    if (pid == 0) {
        // Child process
        printf("Child Process: PID = %d, Parent PID = %d\n", getpid(), getppid());
    } else if (pid > 0) {
        // Parent process
        printf("Parent Process: PID = %d, Child PID = %d\n", getpid(), pid);
    } else {
        // Fork failed
        printf("Fork failed!\n");
    }

    return 0;
}
```

Program 2: Preemptive Priority Scheduling Algorithm

```
#include <stdio.h>
#include <limits.h>

struct process {
    int pid;
    int burst_time;
    int priority;
    int remaining_time;
    int waiting_time;
    int turnaround_time;
};

void calculate_priority(struct process p[], int n) {
    int time = 0, completed = 0;
    int highest_priority = 0, min_time = INT_MAX;
```

```
    int finish_time;
    int check = 0;

    while (completed != n) {
        for (int i = 0; i < n; i++) {
            if (p[i].remaining_time > 0 && p[i].priority < min_time) {
                min_time = p[i].priority;
                highest_priority = i;
                check = 1;
            }
        }

        if (check == 0) {
            time++;
            continue;
        }

        p[highest_priority].remaining_time--;
        min_time = p[highest_priority].priority;

        if (p[highest_priority].remaining_time == 0) {
            completed++;
            check = 0;

            finish_time = time + 1;
            p[highest_priority].turnaround_time = finish_time;
            p[highest_priority].waiting_time = p[highest_priority].turnaround_time - p[highest_priority].burst_time;

            if (p[highest_priority].waiting_time < 0) {
                p[highest_priority].waiting_time = 0;
            }
        }
        time++;
    }
}

void display_priority(struct process p[], int n) {
    int total_waiting = 0, total_turnaround = 0;
    printf("\nPID\tPriority\tBurst Time\tWaiting Time\tTurnaround Time\n");
```

```
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\t\t%d\n", p[i].pid, p[i].priority, p[i].burst_time, p[i].waiting_time, p[i].turnaround_time);
        total_waiting += p[i].waiting_time;
        total_turnaround += p[i].turnaround_time;
    }

    printf("\nAverage Waiting Time: %.2f", (float)total_waiting / n);
    printf("\nAverage Turnaround Time: %.2f", (float)total_turnaround / n);
}

int main() {
    int n;
    printf("Enter number of processes: ");
    scanf("%d", &n);

    struct process p[n];
    for (int i = 0; i < n; i++) {
        p[i].pid = i + 1;
        printf("Enter burst time for process %d: ", p[i].pid);
        scanf("%d", &p[i].burst_time);
        printf("Enter priority for process %d: ", p[i].pid);
        scanf("%d", &p[i].priority);
        p[i].remaining_time = p[i].burst_time;
    }

    calculate_priority(p, n);
    display_priority(p, n);

    return 0;
}
```