Slip 21

Program 1: Use of `nice()` System Call

```
#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>


int main() {

   pid_t pid = fork();


   if (pid == 0) {

      // Child process

      printf("Child Process: PID = %d, Default priority = %d\n", getpid(), getpriority(PRIO_PROCESS, 0));


      // Change priority using nice()

      nice(5);

      printf("Child Process: PID = %d, New priority = %d\n", getpid(), getpriority(PRIO_PROCESS, 0));

   }

   else if (pid > 0) {

      // Parent process

      wait(NULL);  // Wait for the child process to complete

      printf("Parent Process: PID = %d\n", getpid());

   }

   else {

      // Fork failed

      printf("Fork failed!\n");

   }


   return 0;

}
```

Program 2: Non-preemptive Priority Scheduling

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

```c
struct process {
    int pid;
    int burst_time;
    int priority;
    int waiting_time;
    int turnaround_time;
};

void calculate_priority(struct process p[], int n) {
    int total_waiting = 0, total_turnaround = 0;

    p[0].waiting_time = 0;
    for (int i = 1; i < n; i++) {
        p[i].waiting_time = p[i-1].waiting_time + p[i-1].burst_time;
    }

    for (int i = 0; i < n; i++) {
        p[i].turnaround_time = p[i].waiting_time + p[i].burst_time;
        total_waiting += p[i].waiting_time;
        total_turnaround += p[i].turnaround_time;
    }

    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);
    }

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

void sort_by_priority(struct process p[], int n) {
    struct process temp;
    for (int i = 0; i < n-1; i++) {
        for (int j = i+1; j < n; j++) {
            if (p[i].priority > p[j].priority) {
                temp = p[i];
                p[i] = p[j];
                p[j] = temp;
            }
        }
    }
}

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);
    }

    sort_by_priority(p, n);
    calculate_priority(p, n);

    return 0;
}
```