Slip 15

Program 1: Shell with `count` Command

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

void count_lines(char *filename) {
    FILE *file = fopen(filename, "r");
    if (file == NULL) {
        printf("File %s not found.\n", filename);
        return;
    }

    int lines = 0;
    char ch;
    while ((ch = fgetc(file)) != EOF) {
        if (ch == '\n') lines++;
    }

    printf("Total lines: %d\n", lines);
    fclose(file);
}

void count_words(char *filename) {
    FILE *file = fopen(filename, "r");
    if (file == NULL) {
        printf("File %s not found.\n", filename);
        return;
    }

    int words = 0;
    char word[100];
    while (fscanf(file, "%s", word) != EOF) {
        words++;
```

```
    }

    printf("Total words: %d\n", words);
    fclose(file);
}

void count_chars(char *filename) {
    FILE *file = fopen(filename, "r");
    if (file == NULL) {
        printf("File %s not found.\n", filename);
        return;
    }

    int chars = 0;
    char ch;
    while ((ch = fgetc(file)) != EOF) {
        chars++;
    }

    printf("Total characters: %d\n", chars);
    fclose(file);
}

int main() {
    char command[100], *args[10];
    while (1) {
        printf("\nmyshell$ ");
        fgets(command, 100, stdin);
        command[strlen(command) - 1] = '\0';  // Remove newline

        char *token = strtok(command, " ");
        int i = 0;
        while (token != NULL) {
            args[i++] = token;
            token = strtok(NULL, " ");
```

```
    }
    args[i] = NULL;

    if (strcmp(args[0], "count") == 0) {
        if (strcmp(args[1], "l") == 0) {
            count_lines(args[2]);
        } else if (strcmp(args[1], "w") == 0) {
            count_words(args[2]);
        }
```

Program 2: Non-preemptive Priority Scheduling

```
#include <stdio.h>

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;
}
```