Slip 14

Program 1: Shell with `list` Command

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <dirent.h>


void list(char *option, char *dirname) {

    DIR *dir;

    struct dirent *entry;

    dir = opendir(dirname);

    if (dir == NULL) {

        printf("Directory %s not found.\n", dirname);

        return;

    }


    if (strcmp(option, "f") == 0) {

        while ((entry = readdir(dir)) != NULL) {

            if (entry->d_type == DT_REG) {

                printf("%s\n", entry->d_name);

            }

        }

    } else if (strcmp(option, "n") == 0) {

        int dc = 0, fc = 0;

        while ((entry = readdir(dir)) != NULL) {

            if (entry->d_type == DT_DIR) dc++;

            if (entry->d_type == DT_REG) fc++;

        }

        printf("%d Dir(s)\t%d File(s)\n", dc, fc);

    }


    closedir(dir);

}
```

```
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], "list") == 0) {
            list(args[1], args[2]);
        } else if (strcmp(args[0], "exit") == 0) {
            exit(0);
        } else {
            int pid = fork();
            if (pid == 0) {
                execvp(args[0], args);
                exit(0);
            } else {
                wait(NULL);
            }
        }
    }
    return 0;
}
```

Program 2: Preemptive Shortest Job First (SJF) Scheduling

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

```
struct process {
    int pid;
    int burst_time;
    int arrival_time;
    int remaining_time;
    int waiting_time;
    int turnaround_time;
};

void calculate_sjf(struct process p[], int n) {
    int time = 0, completed = 0;
    int shortest = 0, min_time = INT_MAX;
    int finish_time;
    int check = 0;

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

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

        p[shortest].remaining_time--;
        min_time = p[shortest].remaining_time;

        if (min_time == 0) {
            min_time = INT_MAX;
        }

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

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

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

void display_sjf(struct process p[], int n) {
    int total_waiting = 0, total_turnaround = 0;
    printf("\nPID\tArrival Time\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].arrival_time, 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 arrival time for process %d: ", p[i].pid);
        scanf("%d", &p[i].arrival_time);
        printf("Enter burst time for process %d: ", p[i].pid);
        scanf("%d", &p[i].burst_time);
        p[i].remaining_time = p[i].burst_time;
    }

    calculate_sjf(p, n);
    display_sjf(p, n);

    return 0;
}
```