Slip 11

Program 1: LRU Page Replacement Algorithm

```
#include <stdio.h>

#define MAX 20


int frames[MAX], ref[MAX], mem[MAX][MAX], time[MAX], faults = 0, m, n, counter = 0;


void accept() {

    printf("Enter number of frames: ");

    scanf("%d", &n);

    printf("Enter number of references: ");

    scanf("%d", &m);

    printf("Enter reference string:\n");

    for (int i = 0; i < m; i++) {

        printf("[%d] = ", i);

        scanf("%d", &ref[i]);

    }

}


int search(int pno) {

    for (int i = 0; i < n; i++) {

        if (frames[i] == pno) return i;

    }

    return -1;

}


int get_lru() {

    int min = 9999, min_i = 0;

    for (int i = 0; i < n; i++) {

        if (time[i] < min) {

            min = time[i];

            min_i = i;

        }

    }
    return min_i;
}

void lru() {
    for (int i = 0; i < m; i++) {
        int k = search(ref[i]);
        if (k == -1) {
            if (counter < n) {
                frames[counter] = ref[i];
                time[counter] = i;
                counter++;
            } else {
                int pos = get_lru();
                frames[pos] = ref[i];
                time[pos] = i;
            }
            faults++;
        } else {
            time[k] = i;
        }
        for (int j = 0; j < n; j++) {
            mem[j][i] = frames[j];
        }
    }
}

void disp() {
    printf("\nReference String:\n");
    for (int i = 0; i < m; i++) {
        printf("%3d", ref[i]);
    }
    printf("\n\nFrame Allocation:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (mem[i][j]) {
                printf("%3d", mem[i][j]);
            } else {
                printf("   ");
            }
        }
        printf("\n");
    }
    printf("\nTotal Page Faults: %d\n", faults);
}

int main() {
    accept();
    lru();
    disp();
    return 0;
}
```

Program 2

```
#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;
}
```