Slip 7

Program 1: Optimal Page Replacement Algorithm

```c
#include <stdio.h>
#define MAX 20

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

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 predict(int current_index) {
    int farthest = current_index, pos = -1;
    for (int i = 0; i < n; i++) {
        int j;
        for (j = current_index; j < m; j++) {
            if (frames[i] == ref[j]) {
                if (j > farthest) {
                    farthest = j;
```

```c
                pos = i;
            }
            break;
        }
    }
    if (j == m) return i;
}
return (pos == -1) ? 0 : pos;
}


void optimal_page_replacement() {
    for (int i = 0; i < m; i++) {
        if (search(ref[i]) == -1) {
            if (i < n) {
                frames[i] = ref[i];
            } else {
                int pos = predict(i + 1);
                frames[pos] = ref[i];
            }
            faults++;
        }
        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++) {
```

```c
        if (mem[i][j]) {

            printf("%3d", mem[i][j]);

        } else {

            printf("   ");

        }

    }

    printf("\n");

  }

  printf("\nTotal Page Faults: %d\n", faults);

}


int main() {

  accept();

  optimal_page_replacement();

  disp();

  return 0;

}
```

Program 2: Shell with `search` Command

```c
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


void search_file(char *option, char *filename, char *pattern) {

  FILE *file = fopen(filename, "r");

  if (file == NULL) {

    printf("File %s not found.\n", filename);

    return;

  }


  char line[256];

  int count = 0, line_num = 0;


  while (fgets(line, sizeof(line), file)) {

    line_num++;
```

```c
        if (strstr(line, pattern) != NULL) {

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

                printf("Line %d: %s", line_num, line);

            }

            count++;

        }

    }


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

        printf("Pattern '%s' occurred %d times in file %s.\n", pattern, count, filename);

    }


    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], "search") == 0) {

            search_file(args[1], args[2], args[3]);

        } else if (strcmp(args[0], "exit") == 0) {

            exit(0);

        } else {
```

```c
        int pid = fork();

        if (pid == 0) {

            execvp(args[0], args);

            exit(0);

        } else {

            wait(NULL);

        }

    }

}

return 0;

}
```