Slip 16

Program 1: Optimal Page Replacement Algorithm

```
#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;
```

```
                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++) {
```

```
            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: FCFS Scheduling Algorithm

```
#include <stdio.h>

struct process {
    int pid;
    int burst_time;
    int waiting_time;
    int turnaround_time;
};

void calculate_fcfs(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\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\n", p[i].pid, 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);
}

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);
    }

    calculate_fcfs(p, n);

    return 0;
}
```