ASSIGNMENT 9
21/11/23

NAME : SHRESTH SONKAR
REGNO : 20214272
GROUP : CS5D
TOPIC : OS LAB
CODE : CS-15203

```c
/*
 * Q1 :
 * Write a C program that is passed a virtual address
on the CLI
 * Output the page number and offset for the given
address
 */

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

#define PAGE_SIZE 4096

void getPageNumberAndOffset(unsigned int address) {
    unsigned int pageNumber = address / PAGE_SIZE;
    unsigned int offset = address % PAGE_SIZE;

    printf("The address %u contains : \n", address);
    printf("page number = %u\n", pageNumber);
    printf("offset = %u\n", offset);
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        printf("Usage: %s <virtual_address>\n",
argv[0]);
        return 1;
    }

    unsigned int virtualAddress = atoi(argv[1]);
    getPageNumberAndOffset(virtualAddress);

    return 0;
}
```

```c
/*
 * Q2 :
 * Write a program that implements the FIFO and LRU
page-replacement algorithms.
 * First, generate a random page reference string where
page numbers range from 0 to 9.
 * Apply the random page-reference string to each
algorithm, and record the number of page faults
incurred.
 * Implement the replacement algorithms so that the
number of page frames can vary from 1 to 7.
 * Assume that demand paging is used.
 */

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <time.h>

bool pageExists(int page, int *frames, int numFrames) {
    for (int i = 0; i < numFrames; ++i) {
        if (frames[i] == page) {
            return true;
        }
    }
    return false;
}

int fifoPageReplacement(int *pages, int numPages, int
numFrames) {
    int *frames = (int *) malloc(numFrames *
sizeof(int));
    int pageFaults = 0;
    int frameIndex = 0;

    for (int i = 0; i < numPages; ++i) {
        if (!pageExists(pages[i], frames, numFrames)) {
            frames[frameIndex] = pages[i];
            frameIndex = (frameIndex + 1) % numFrames;
            pageFaults++;
        }
    }

    free(frames);
    return pageFaults;
```

```
}

int lruPageReplacement(int *pages, int numPages, int
numFrames) {
    int *frames = (int *) malloc(numFrames *
sizeof(int));
    int *counter = (int *) malloc(numFrames *
sizeof(int));
    int pageFaults = 0;

    for (int i = 0; i < numFrames; ++i) {
        frames[i] = -1;
        counter[i] = 0;
    }

    for (int i = 0; i < numPages; ++i) {
        int j;
        for (j = 0; j < numFrames; ++j) {
            if (frames[j] == pages[i]) {
                counter[j] = i + 1;
                break;
            }
        }

        if (j == numFrames) {
            int leastUsed = 0;

            for (int k = 1; k < numFrames; ++k) {
                if (counter[k] < counter[leastUsed])
                    leastUsed = k;
            }

            frames[leastUsed] = pages[i];
            counter[leastUsed] = i + 1;
            pageFaults++;
        }
    }

    free(frames);
    free(counter);
    return pageFaults;
}

int main() {
    srand(time(NULL));
```

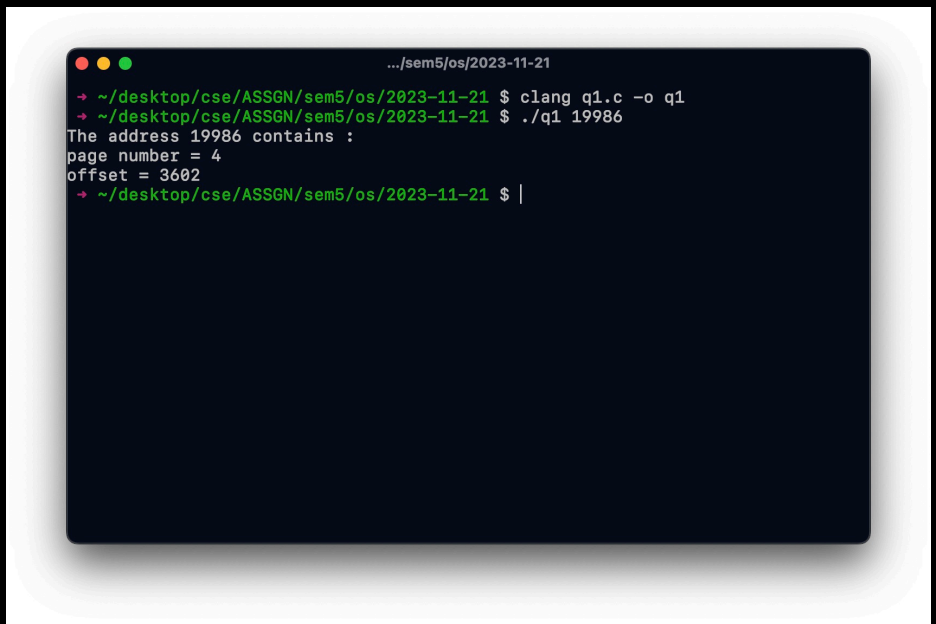
```
    int numPages = 30;
    int pages[numPages];

    for (int i = 0; i < numPages; ++i)
        pages[i] = rand() % 10;

    printf("Page reference string: ");
    for (int i = 0; i < numPages; ++i)
        printf("%d ", pages[i]);
    printf("\n");

    for (int numFrames = 1; numFrames <= 7; +
+numFrames) {
        int fifoFaults = fifoPageReplacement(pages,
numPages, numFrames);
        int lruFaults = lruPageReplacement(pages,
numPages, numFrames);

        printf("\nNumber of frames: %d\n", numFrames);
        printf("FIFO Page Faults: %d\n", fifoFaults);
        printf("LRU Page Faults: %d\n", lruFaults);
    }

    return 0;
}
```

```c
/*
 * Q3a :
 * The producer process will generate the Catalan
sequence and write it to a shared memory object.
 */

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define MAX_SEQUENCE 100
#define SHARED_MEM_SIZE (MAX_SEQUENCE * sizeof(unsigned
long long))

unsigned long long catalan[MAX_SEQUENCE];

void *produceCatalan(void *arg) {
    int n = *((int *) arg);

    catalan[0] = 1;
    for (int i = 1; i < n; i++) {
        catalan[i] = 0;
        for (int j = 0; j < i; j++) {
            catalan[i] += catalan[j] * catalan[i - j -
1];
        }
    }

    key_t key = ftok("shmfile", 65);
    int shmid = shmget(key, SHARED_MEM_SIZE, IPC_CREAT
| 0666);
    if (shmid < 0) {
        perror("shmget");
        exit(1);
    }

    unsigned long long *shared_mem = (unsigned long
long *) shmat(shmid, NULL, 0);
    if (shared_mem == (void *) -1) {
        perror("shmat");
        exit(1);
```

```
    }

    for (int i = 0; i < n; i++)
        shared_mem[i] = catalan[i];
    shmdt(shared_mem);
    pthread_exit(NULL);
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        printf("Usage: %s
<number_of_catalan_numbers>\n", argv[0]);
        return 1;
    }

    int n = atoi(argv[1]);
    pthread_t producer_thread;

    pthread_create(&producer_thread, NULL,
produceCatalan, &n);
    pthread_join(producer_thread, NULL);

    printf("Producer has generated the first %d Catalan
numbers.\n", n);
    return 0;
}
```

```c
/*
 * Q3a :
 * The producer process will generate the Catalan
sequence and write it to a shared memory object.
 */

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define MAX_SEQUENCE 100
#define SHARED_MEM_SIZE (MAX_SEQUENCE * sizeof(unsigned
long long))

unsigned long long catalan[MAX_SEQUENCE];

void *produceCatalan(void *arg) {
    int n = *((int *) arg);

    catalan[0] = 1;
    for (int i = 1; i < n; i++) {
        catalan[i] = 0;
        for (int j = 0; j < i; j++) {
            catalan[i] += catalan[j] * catalan[i - j -
1];
        }
    }

    key_t key = ftok("shmfile", 65);
    int shmid = shmget(key, SHARED_MEM_SIZE, IPC_CREAT
| 0666);
    if (shmid < 0) {
        perror("shmget");
        exit(1);
    }

    unsigned long long *shared_mem = (unsigned long
long *) shmat(shmid, NULL, 0);
    if (shared_mem == (void *) -1) {
        perror("shmat");
        exit(1);
```

```c
    }

    for (int i = 0; i < n; i++)
        shared_mem[i] = catalan[i];
    shmdt(shared_mem);
    pthread_exit(NULL);
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        printf("Usage: %s
<number_of_catalan_numbers>\n", argv[0]);
        return 1;
    }

    int n = atoi(argv[1]);
    pthread_t producer_thread;

    pthread_create(&producer_thread, NULL,
produceCatalan, &n);
    pthread_join(producer_thread, NULL);

    printf("Producer has generated the first %d Catalan
numbers.\n", n);
    return 0;
}
```



```
.../sem5/os/2023-11-21
➜ ~/desktop/cse/ASSGN/sem5/os/2023-11-21 $ clang q2.c -o q2
➜ ~/desktop/cse/ASSGN/sem5/os/2023-11-21 $ ./q2
Page reference string: 6 3 1 1 0 7 3 4 7 6 3 0 0 4 3 9 8 2 8 3 6 8 4 1 3 2 8 2
 5 4

Number of frames: 1
FIFO Page Faults: 28
LRU Page Faults: 28

Number of frames: 2
FIFO Page Faults: 26
LRU Page Faults: 26

Number of frames: 3
FIFO Page Faults: 23
LRU Page Faults: 23

Number of frames: 4
FIFO Page Faults: 19
LRU Page Faults: 20

Number of frames: 5
FIFO Page Faults: 16
LRU Page Faults: 16

Number of frames: 6
FIFO Page Faults: 13
LRU Page Faults: 12

Number of frames: 7
FIFO Page Faults: 11
LRU Page Faults: 11
➜ ~/desktop/cse/ASSGN/sem5/os/2023-11-21 $
```

```
.../sem5/os/2023-11-21
➜ ~/desktop/cse/ASSGN/sem5/os/2023-11-21 $ clang q3a.c -o q3a
➜ ~/desktop/cse/ASSGN/sem5/os/2023-11-21 $ ./q3a 5
Producer has generated the first 5 Catalan numbers.
➜ ~/desktop/cse/ASSGN/sem5/os/2023-11-21 $
```

```
➜ ~/desktop/cse/ASSGN/sem5/os/2023-11-21 $ clang q3b.c -o q3b
➜ ~/desktop/cse/ASSGN/sem5/os/2023-11-21 $ ./q3b 5
Consumer reading 5 Catalan numbers from shared memory:
1 1 2 5 14
➜ ~/desktop/cse/ASSGN/sem5/os/2023-11-21 $
```