SHREERAM 20214033

OPERATING SYSTEM LAB
ASSIGNMENT 9

1. Assume that a system has a 32-bit virtual address with a 4-KB page size. Write a C program that is passed a virtual address (in decimal) on the command line and have it output the page number and offset for the given address. As an example, your program would run as follows: ./a.out 19986 Your program would output: The address 19986 contains: page number = 4 offset = 3602 Writing this program will require using the appropriate data type to store 32 bits. We encourage you to use unsigned data types as well.

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

#define PAGE_SIZE 4096 // 4 KB page size
 void calculatePageAndOffset(unsigned int
virtualAddress);
 int main(int argc, char *argv[]) {     if (argc != 2) {
fprintf(stderr, "Usage: %s <virtual_address>\n", argv[0]);
return EXIT_FAILURE;
    }
    unsigned int virtualAddress = atoi(argv[1]);
calculatePageAndOffset(virtualAddress);
     return
EXIT_SUCCESS;
}  void calculatePageAndOffset(unsigned int
virtualAddress) {
    // Assuming a 32-bit virtual address     unsigned
int pageNumber = virtualAddress / PAGE_SIZE;
unsigned int offset = virtualAddress % PAGE_SIZE;




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

2. Write a program that implements the FIFO and LRU pagereplacement 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 by each algorithm. Implement the replacement algorithms so that the number of page frames can vary from 1 to 7. Assume that demand paging is used.

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

// Function declarations void generatePageReferences(int
pageReferences[], int size); int fifoPageReplacement(int
pageReferences[], int size, int numFrames); int lruPageReplacement(int
pageReferences[], int size, int numFrames);
 int main() {     const int referenceStringSize = 20;  // Adjust the
size as needed     int pageReferences[referenceStringSize];

    // Generate a random page reference string
    generatePageReferences(pageReferences, referenceStringSize);

    printf("Page Reference String: ");
    for  (int  i  =  0;  i  <  referenceStringSize;  ++i)  {
printf("%d ", pageReferences[i]);
    }
printf("\n\n");

    // Test FIFO and LRU with different numbers of page frames
for (int numFrames = 1; numFrames <= 7; ++numFrames) {
printf("Number of Frames: %d\n", numFrames);
```

```
        int fifoFaults = fifoPageReplacement(pageReferences,
referenceStringSize, numFrames);              printf("FIFO Page Faults: %d\n",
fifoFaults);
        int lruFaults = lruPageReplacement(pageReferences,
referenceStringSize, numFrames);              printf("LRU Page Faults: %d\n",
lruFaults);

printf("\n");
    }
return 0;
}  void generatePageReferences(int pageReferences[], int size) {
srand(42); // Seed for reproducibility     for (int i = 0; i < size; ++i)
{         pageReferences[i] = rand() % 10;  // Page numbers range from 0
to 9
    }
}  int fifoPageReplacement(int pageReferences[], int size, int
numFrames) {     int pageFrames[numFrames];     int pageFaults = 0;
int frameIndex = 0;
     for (int i = 0; i < numFrames; ++i) {         pageFrames[i] = -1; //
Initialize page frames to -1 (invalid page number)     }
    for (int i = 0; i < size; ++i) {
bool pageFault = true;

        // Check if the page is already in a frame
for (int j = 0; j < numFrames; ++j) {             if
(pageFrames[j] == pageReferences[i]) {
pageFault = false;                 break;
            }
        }

        // If there is a page fault, replace the oldest
page         if (pageFault) {
pageFrames[frameIndex] = pageReferences[i];
```

```
            frameIndex = (frameIndex + 1) % numFrames;
            ++pageFaults;
        }

        // Print the current state of page frames
(optional)         printf("Page Frames (FIFO): ");
for (int j = 0; j < numFrames; ++j) {
printf("%d ", pageFrames[j]);
        }
printf("\n");
    }      return
pageFaults;
}  int lruPageReplacement(int pageReferences[], int size, int
numFrames) {     int pageFrames[numFrames];     int pageFaults = 0;
int age[numFrames];
     for (int i = 0; i < numFrames; ++i) {         pageFrames[i] = -1; //
Initialize page frames to -1 (invalid page number)         age[i] = 0;
    }      for (int i = 0; i <
size; ++i) {         bool pageFault
= true;

        // Check if the page is already in a frame
for (int j = 0; j < numFrames; ++j) {             if
(pageFrames[j] == pageReferences[i]) {
pageFault = false;
                age[j] = 0; // Reset the age of the accessed page
            } else {
age[j]++;
            }
        }

        // If there is a page fault, replace the least recently used
page          if (pageFault) {             int lruIndex = 0;
            for (int j = 1; j < numFrames; ++j) {
if (age[j] > age[lruIndex]) {
lruIndex = j;
                }
```

```
            }
             pageFrames[lruIndex] =
pageReferences[i];             age[lruIndex] = 0;
            ++pageFaults;
        }

        // Print the current state of page frames (optional)
printf("Page Frames (LRU): ");         for (int j = 0; j <
numFrames; ++j) {             printf("%d ", pageFrames[j]);
        }
printf("\n");
    }      return
pageFaults;
}
```

OUTPUT:

```
Page Reference String: 6 0 1 1 2 8 1 0 5 3 4 3 7 4 6 2 2 8 8 9

Number of Frames: 1
Page Frames (FIFO): 6
Page Frames (FIFO): 0
Page Frames (FIFO): 1
Page Frames (FIFO): 1
Page Frames (FIFO): 2
Page Frames (FIFO): 8
Page Frames (FIFO): 1
Page Frames (FIFO): 0
Page Frames (FIFO): 5
Page Frames (FIFO): 3
Page Frames (FIFO): 4
Page Frames (FIFO): 3
Page Frames (FIFO): 7
Page Frames (FIFO): 4
Page Frames (FIFO): 6
```

```
Page Frames (FIFO): 2
Page Frames (FIFO): 2
Page Frames (FIFO): 8
Page Frames (FIFO): 8
Page Frames (FIFO): 9
FIFO Page Faults: 17
Page Frames (LRU): 6
Page Frames (LRU): 0
Page Frames (LRU): 1
Page Frames (LRU): 1
Page Frames (LRU): 2
Page Frames (LRU): 8
Page Frames (LRU): 1
Page Frames (LRU): 0
Page Frames (LRU): 5
Page Frames (LRU): 3
Page Frames (LRU): 4
Page Frames (LRU): 3
Page Frames (LRU): 7
Page Frames (LRU): 4
Page Frames (LRU): 6
Page Frames (LRU): 2
Page Frames (LRU): 2
Page Frames (LRU): 8
Page Frames (LRU): 8
Page Frames (LRU): 9
LRU Page Faults: 17

Number of Frames: 2
Page Frames (FIFO): 6 -1
Page Frames (FIFO): 6 0
Page Frames (FIFO): 1 0
Page Frames (FIFO): 1 0
Page Frames (FIFO): 1 2
Page Frames (FIFO): 8 2
Page Frames (FIFO): 8 1
Page Frames (FIFO): 0 1
Page Frames (FIFO): 0 5
```

```
Page Frames (FIFO): 3 5
Page Frames (FIFO): 3 4
Page Frames (FIFO): 3 4
Page Frames (FIFO): 7 4
Page Frames (FIFO): 7 4
Page Frames (FIFO): 7 6
Page Frames (FIFO): 2 6
Page Frames (FIFO): 2 6
Page Frames (FIFO): 2 8
Page Frames (FIFO): 2 8
Page Frames (FIFO): 9 8
FIFO Page Faults: 15
Page Frames (LRU): 6 -1
Page Frames (LRU): 6 0
Page Frames (LRU): 1 0
Page Frames (LRU): 1 0
Page Frames (LRU): 1 2
Page Frames (LRU): 8 2
Page Frames (LRU): 8 1
```

```
Page Frames (LRU): 0 1
Page Frames (LRU): 0 5
Page Frames (LRU): 3 5
Page Frames (LRU): 3 4
Page Frames (LRU): 3 4
Page Frames (LRU): 3 7
Page Frames (LRU): 4 7
Page Frames (LRU): 4 6
Page Frames (LRU): 2 6
Page Frames (LRU): 2 6
Page Frames (LRU): 2 8
Page Frames (LRU): 2 8
Page Frames (LRU): 9 8
LRU Page Faults: 16

Number of Frames: 3
Page Frames (FIFO): 6 -1 -1
Page Frames (FIFO): 6 0 -1
Page Frames (FIFO): 6 0 1
```

```
Page Frames (FIFO): 6 0 1
Page Frames (FIFO): 2 0 1
Page Frames (FIFO): 2 8 1
Page Frames (FIFO): 2 8 1
Page Frames (FIFO): 2 8 0
Page Frames (FIFO): 5 8 0
Page Frames (FIFO): 5 3 0
Page Frames (FIFO): 5 3 4
Page Frames (FIFO): 5 3 4
Page Frames (FIFO): 7 3 4
Page Frames (FIFO): 7 3 4
Page Frames (FIFO): 7 6 4
Page Frames (FIFO): 7 6 2
Page Frames (FIFO): 7 6 2
Page Frames (FIFO): 8 6 2
Page Frames (FIFO): 8 6 2
Page Frames (FIFO): 8 9 2
FIFO Page Faults: 14
Page Frames (LRU): 6 -1 -1
```

```
Page Frames (LRU): 6 0 -1
Page Frames (LRU): 6 0 1
Page Frames (LRU): 6 0 1
Page Frames (LRU): 2 0 1
Page Frames (LRU): 2 8 1
Page Frames (LRU): 2 8 1 Page Frames (LRU): 0 8 1
Page Frames (LRU): 0 5 1
Page Frames (LRU): 0 5 3
Page Frames (LRU): 4 5 3
Page Frames (LRU): 4 5 3
Page Frames (LRU): 4 7 3
Page Frames (LRU): 4 7 3
Page Frames (LRU): 4 7 6
Page Frames (LRU): 4 2 6
Page Frames (LRU): 4 2 6
Page Frames (LRU): 8 2 6
Page Frames (LRU): 8 2 6
Page Frames (LRU): 8 2 9
LRU Page Faults: 14
```

```
Number of Frames: 4
Page Frames (FIFO): 6 -1 -1 -1
Page Frames (FIFO): 6 0 -1 -1
Page Frames (FIFO): 6 0 1 -1
Page Frames (FIFO): 6 0 1 -1
Page Frames (FIFO): 6 0 1 2
Page Frames (FIFO): 8 0 1 2
Page Frames (FIFO): 8 0 1 2
Page Frames (FIFO): 8 0 1 2
Page Frames (FIFO): 8 5 1 2
Page Frames (FIFO): 8 5 3 2
Page Frames (FIFO): 8 5 3 4
Page Frames (FIFO): 8 5 3 4
Page Frames (FIFO): 7 5 3 4
Page Frames (FIFO): 7 5 3 4
Page Frames (FIFO): 7 6 3 4
Page Frames (FIFO): 7 6 2 4
Page Frames (FIFO): 7 6 2 4
Page Frames (FIFO): 7 6 2 8
Page Frames (FIFO): 7 6 2 8
Page Frames (FIFO): 9 6 2 8
FIFO Page Faults: 13
Page Frames (LRU): 6 -1 -1 -1
Page Frames (LRU): 6 0 -1 -1
Page Frames (LRU): 6 0 1 -1
Page Frames (LRU): 6 0 1 -1
Page Frames (LRU): 6 0 1 2
Page Frames (LRU): 8 0 1 2
Page Frames (LRU): 8 0 1 2
Page Frames (LRU): 8 0 1 2
Page Frames (LRU): 8 0 1 5
Page Frames (LRU): 3 0 1 5
Page Frames (LRU): 3 0 4 5
Page Frames (LRU): 3 0 4 5
Page Frames (LRU): 3 7 4 5
Page Frames (LRU): 3 7 4 5
Page Frames (LRU): 3 7 4 6
Page Frames (LRU): 2 7 4 6
Page Frames (LRU): 2 7 4 6
Page Frames (LRU): 2 8 4 6
Page Frames (LRU): 2 8 4 6
Page Frames (LRU): 2 8 9 6
LRU Page Faults: 13

Number of Frames: 5
Page Frames (FIFO): 6 -1 -1 -1 -1
Page Frames (FIFO): 6 0 -1 -1 -1
Page Frames (FIFO): 6 0 1 -1 -1
Page Frames (FIFO): 6 0 1 -1 -1
Page Frames (FIFO): 6 0 1 2 -1
Page Frames (FIFO): 6 0 1 2 8
Page Frames (FIFO): 6 0 1 2 8
Page Frames (FIFO): 6 0 1 2 8
Page Frames (FIFO): 5 0 1 2 8
Page Frames (FIFO): 5 3 1 2 8
Page Frames (FIFO): 5 3 4 2 8
Page Frames (FIFO): 5 3 4 7 8
Page Frames (FIFO): 5 3 4 7 8
Page Frames (FIFO): 5 3 4 7 6
Page Frames (FIFO): 2 3 4 7 6
Page Frames (FIFO): 2 3 4 7 6
Page Frames (FIFO): 2 8 4 7 6
Page Frames (FIFO): 2 8 4 7 6
Page Frames (FIFO): 2 8 9 7 6
FIFO Page Faults: 13
Page Frames (LRU): 6 -1 -1 -1 -1
Page Frames (LRU): 6 0 -1 -1 -1
Page Frames (LRU): 6 0 1 -1 -1
Page Frames (LRU): 6 0 1 -1 -1
Page Frames (LRU): 6 0 1 2 -1
Page Frames (LRU): 6 0 1 2 8
Page Frames (LRU): 6 0 1 2 8
Page Frames (LRU): 6 0 1 2 8
Page Frames (LRU): 5 0 1 2 8
```

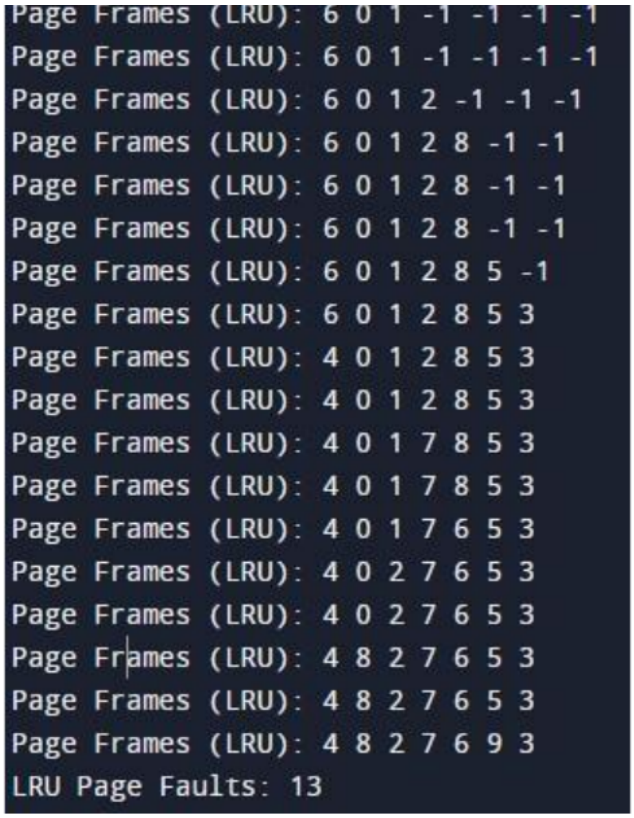
```
Page Frames (LRU): 5 0 1 3 8
Page Frames (LRU): 5 0 1 3 4
Page Frames (LRU): 5 0 1 3 4
Page Frames (LRU): 5 0 7 3 4
Page Frames (LRU): 5 0 7 3 4
Page Frames (LRU): 5 6 7 3 4
Page Frames (LRU): 2 6 7 3 4
Page Frames (LRU): 2 6 7 3 4
Page Frames (LRU): 2 6 7 8 4
Page Frames (LRU): 2 6 7 8 4
Page Frames (LRU): 2 6 9 8 4
LRU Page Faults: 13

Number of Frames: 6
Page Frames (FIFO): 6 -1 -1 -1 -1 -1
Page Frames (FIFO): 6 0 -1 -1 -1 -1
Page Frames (FIFO): 6 0 1 -1 -1 -1
Page Frames (FIFO): 6 0 1 -1 -1 -1
Page Frames (FIFO): 6 0 1 2 -1 -1
```

```
Page Frames (FIFO): 6 0 1 2 8 -1
Page Frames (FIFO): 6 0 1 2 8 -1
Page Frames (FIFO): 6 0 1 2 8 -1
Page Frames (FIFO): 6 0 1 2 8 5
Page Frames (FIFO): 3 0 1 2 8 5
Page Frames (FIFO): 3 4 1 2 8 5
Page Frames (FIFO): 3 4 1 2 8 5
Page Frames (FIFO): 3 4 7 2 8 5
Page Frames (FIFO): 3 4 7 2 8 5
Page Frames (FIFO): 3 4 7 6 8 5
Page Frames (FIFO): 3 4 7 6 2 5
Page Frames (FIFO): 3 4 7 6 2 5
Page Frames (FIFO): 3 4 7 6 2 8
Page Frames (FIFO): 3 4 7 6 2 8
Page Frames (FIFO): 9 4 7 6 2 8
FIFO Page Faults: 13
Page Frames (LRU): 6 -1 -1 -1 -1 -1
Page Frames (LRU): 6 0 -1 -1 -1 -1
Page Frames (LRU): 6 0 1 -1 -1 -1
```

```
Page Frames (LRU): 6 0 1 -1 -1 -1
Page Frames (LRU): 6 0 1 2 -1 -1
Page Frames (LRU): 6 0 1 2 8 -1
Page Frames (LRU): 6 0 1 2 8 -1
Page Frames (LRU): 6 0 1 2 8 -1
Page Frames (LRU): 6 0 1 2 8 5
Page Frames (LRU): 3 0 1 2 8 5
Page Frames (LRU): 3 0 1 4 8 5
Page Frames (LRU): 3 0 1 4 8 5
Page Frames (LRU): 3 0 1 4 7 5
Page Frames (LRU): 3 0 1 4 7 5
Page Frames (LRU): 3 0 6 4 7 5
Page Frames (LRU): 3 2 6 4 7 5
Page Frames (LRU): 3 2 6 4 7 5
Page Frames (LRU): 3 2 6 4 7 8
Page Frames (LRU): 3 2 6 4 7 8
Page Frames (LRU): 9 2 6 4 7 8
LRU Page Faults: 13
```

```
Number of Frames: 7
Page Frames (FIFO): 6 -1 -1 -1 -1 -1 -1
Page Frames (FIFO): 6 0 -1 -1 -1 -1 -1
Page Frames (FIFO): 6 0 1 -1 -1 -1 -1
Page Frames (FIFO): 6 0 1 -1 -1 -1 -1
Page Frames (FIFO): 6 0 1 2 -1 -1 -1
Page Frames (FIFO): 6 0 1 2 8 -1 -1
Page Frames (FIFO): 6 0 1 2 8 -1 -1
Page Frames (FIFO): 6 0 1 2 8 -1 -1
Page Frames (FIFO): 6 0 1 2 8 5 -1
Page Frames (FIFO): 6 0 1 2 8 5 3
Page Frames (FIFO): 4 0 1 2 8 5 3
Page Frames (FIFO): 4 0 1 2 8 5 3
Page Frames (FIFO): 4 7 1 2 8 5 3
Page Frames (FIFO): 4 7 1 2 8 5 3
Page Frames (FIFO): 4 7 6 2 8 5 3
Page Frames (FIFO): 4 7 6 2 8 5 3
Page Frames (FIFO): 4 7 6 2 8 5 3
Page Frames (FIFO): 4 7 6 2 8 5 3
Page Frames (FIFO): 4 7 6 2 8 5 3
Page Frames (FIFO): 4 7 6 9 8 5 3
FIFO Page Faults: 11
Page Frames (LRU): 6 -1 -1 -1 -1 -1 -1
Page Frames (LRU): 6 0 -1 -1 -1 -1 -1
Page Frames (LRU): 6 0 1 -1 -1 -1 -1
Page Frames (LRU): 6 0 1 -1 -1 -1 -1
Page Frames (LRU): 6 0 1 2 -1 -1 -1
Page Frames (LRU): 6 0 1 2 8 -1 -1
Page Frames (LRU): 6 0 1 2 8 -1 -1
Page Frames (LRU): 6 0 1 2 8 -1 -1
Page Frames (LRU): 6 0 1 2 8 5 -1
Page Frames (LRU): 6 0 1 2 8 5 3
Page Frames (LRU): 4 0 1 2 8 5 3
Page Frames (LRU): 4 0 1 2 8 5 3
Page Frames (LRU): 4 0 1 7 8 5 3
Page Frames (LRU): 4 0 1 7 8 5 3
Page Frames (LRU): 4 0 1 7 6 5 3
Page Frames (LRU): 4 0 2 7 6 5 3
```



3. The Catalan numbers are an integer sequence C_n that appear in tree enumeration problems. The first Catalan numbers for n = 1, 2,

3, ... are 1, 2, 5, 14, 42, 132, ... A formula generating C_n is $C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!n!}$ Design two programs that communicate with shared memory using the Pthread. The producer process will generate the Catalan sequence and write it to a shared memory object. The consumer process will then read and output the sequence from shared memory. In this instance, the producer process will be passed an integer parameter on the command line specifying how many Catalan numbers to produce

(for example, providing 5 on the command line means the producer process will generate the first five Catalan numbers).

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

#define MAX_SEQUENCE_LENGTH 100

// Structure to hold shared data
typedef struct {     int
sequence[MAX_SEQUENCE_LENGTH];
int count;
} SharedData;

// Function declarations
void *producer(void *arg);
void *consumer(void *arg);
 int main(int argc, char *argv[]) {     if (argc != 2) {
fprintf(stderr, "Usage: %s <number_of_catalan_numbers>\n", argv[0]);
return EXIT_FAILURE;
    }      int numCatalanNumbers =
atoi(argv[1]);
    // Create shared memory segment     key_t key =
ftok("/dev/null", 1);     int shmid = shmget(key,
sizeof(SharedData), IPC_CREAT | 0666);     if (shmid == -1) {
perror("shmget");         return EXIT_FAILURE;
    }

    // Attach shared memory segment
    SharedData *sharedData = (SharedData *)shmat(shmid, NULL,
0);     if (sharedData == (SharedData *)(-1)) {
perror("shmat");         return EXIT_FAILURE;
    }

    // Initialize shared data      sharedData-
>count = numCatalanNumbers;
```

```
    // Create producer and consumer threads     pthread_t producerThread,
consumerThread;     pthread_create(&producerThread, NULL, producer, (void
*)sharedData);     pthread_create(&consumerThread, NULL, consumer, (void
*)sharedData);
    // Wait for threads to finish
pthread_join(producerThread, NULL);
pthread_join(consumerThread, NULL);

    // Detach and remove shared memory segment
shmdt(sharedData);     shmctl(shmid, IPC_RMID,
NULL);
     return EXIT_SUCCESS;
}
void *producer(void *arg) {
    SharedData *sharedData = (SharedData *)arg;
     int n = sharedData->count;
int catalanNumber = 1;
     for (int i = 0; i < n; ++i) {         sharedData->sequence[i] =
catalanNumber;         catalanNumber = catalanNumber * 2 * (2 * i +
1) / (i + 2);     }      pthread_exit(NULL);
}
void *consumer(void *arg) {
    SharedData *sharedData = (SharedData *)arg;

    printf("Catalan Sequence: ");
    for (int i = 0; i < sharedData->count; ++i) {       printf("%d
", sharedData->sequence[i]);
    }
    printf("\n");
     pthread_exit(NULL);
}
```