OS Lab Manual

Created by: Himanshu Lodha

1. Process System Calls

Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

int main() {
    pid_t pid;
    pid = fork();

    if (pid == 0) {
        printf("Child process with PID %d\n", getpid());

        // Using execlp to replace the child process image with ls command
        execlp("ls", "ls", "-l", NULL); // Executing ls -l command

        // If execlp fails
        perror("execlp");
        exit(EXIT_FAILURE);
    } else if (pid > 0) {
        printf("Parent process with PID %d\n", getpid());

        int status;
        wait(&status);
        printf("Child process reaped\n");
    } else {
        perror("fork");
        exit(EXIT_FAILURE);
    }
    return 0;
}
```

Output

```
Parent process with PID 62707
Child process with PID 62708
total 20
-rwxrwxrwx 1 bmsit bmsit     0 Feb 26 18:53 abc
-rwxrwxrwx 1 bmsit bmsit 16304 Feb 26 18:53 fork
```

```
-rwxrwxrwx 1 bmsit bmsit   727 Feb 26 18:53 fork.c
Child process reaped
```

2. CPU Scheduling Algorithms

a) FCFS

Code

```c
#include <stdio.h>

int main() {
    int n, bt[20], wt[20], tat[20], avwt = 0, avtat = 0;
    printf("Enter total number of processes (maximum 20): ");
    scanf("%d", &n);

    printf("\nEnter Process Burst Time:\n");
    for (int i = 0; i < n; i++) {
        printf("P[%d]: ", i + 1);
        scanf("%d", &bt[i]);
    }

    wt[0] = 0;
    for (int i = 1; i < n; i++) {
        wt[i] = wt[i - 1] + bt[i - 1];
    }
    printf("\nProcess\tBurst Time\tWaiting Time\tTurnaround Time");
    for (int i = 0; i < n; i++) {
        tat[i] = bt[i] + wt[i];
        avwt += wt[i];
        avtat += tat[i];
        printf("\nP[%d]\t%d\t\t%d\t\t%d", i + 1, bt[i], wt[i], tat[i]);
    }
    avwt /= n;
    avtat /= n;
    printf("\n\nAverage Waiting Time: %d", avwt);
    printf("\nAverage Turnaround Time: %d", avtat);
    return 0;
}
```

Output

```
Enter total number of processes(maximum 20): 4

Enter Process Burst Time:
P[1]: 5
P[2]: 8
```

```
P[3]: 2
P[4]: 7

Process         Burst Time      Waiting Time    Turnaround Time
P[1]            5               0               5
P[2]            8               5               13
P[3]            2               13              15
P[4]            7               15              22

Average Waiting Time:8
Average Turnaround Time:13
```

b) SJF

Code

```c
#include <stdio.h>

int main() {
    int bt[20], p[20], wt[20], tat[20], n, total_wt = 0, total_tat = 0;
    float avg_wt, avg_tat;

    printf("Enter number of processes: ");
    scanf("%d", &n);

    printf("\nEnter Burst Time:\n");
    for (int i = 0; i < n; i++) {
        printf("p%d: ", i + 1);
        scanf("%d", &bt[i]);
        p[i] = i + 1;
    }

    // sort burst time and process number wrt brust time
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (bt[j] > bt[j + 1]) {
                // Swap burst time
                int temp = bt[j];
                bt[j] = bt[j + 1];
                bt[j + 1] = temp;

                // Swap process number
                temp = p[j];
                p[j] = p[j + 1];
                p[j + 1] = temp;
            }
        }
    }

    // Calculate waiting time for each process
    wt[0] = 0;
```

```
    for (int i = 1; i < n; i++) {
        wt[i] = wt[i - 1] + bt[i - 1];
        total_wt += wt[i];
    }

    // Calculate turnaround time for each process
    for (int i = 0; i < n; i++) {
        tat[i] = bt[i] + wt[i];
        total_tat += tat[i];
    }

    // Calculate average waiting time and average turnaround time
    avg_wt = (float)total_wt / n;
    avg_tat = (float)total_tat / n;

    // Print results
    printf("\nProcess\tBurst Time\tWaiting Time\tTurnaround Time");
    for (int i = 0; i < n; i++) {
        printf("\np%d\t\t%d\t\t%d\t\t\t%d", p[i], bt[i], wt[i], tat[i]);
    }
    printf("\n\nAverage Waiting Time=%.2f", avg_wt);
    printf("\nAverage Turnaround Time=%.2f\n", avg_tat);

    return 0;
}
```

Output

```
Enter number of processes: 4

Enter Burst Time:
p1: 4
p2: 5
p3: 8
p4: 2

Process Burst Time      Waiting Time    Turnaround Time
p4              2               0                       2
p1              4               2                       6
p2              5               6                       11
p3              8               11                      19

Average Waiting Time=4.75
Average Turnaround Time=9.50
```

c) Round Robin

Code

```c
#include <stdio.h>

int main()
{
	int n, time_quantum;
	int at[10], bt[10], rt[10], wt[10], tat[10];
	float avg_wt = 0, avg_tat = 0;

	printf("Enter Total Number of Processes: ");
	scanf("%d", &n);

	printf("Enter Arrival Time and Burst Time for Each Process:\n");
	for (int i = 0; i < n; i++)
	{
		printf("Process %d:\n", i + 1);
		printf("Arrival Time: ");
		scanf("%d", &at[i]);
		printf("Burst Time: ");
		scanf("%d", &bt[i]);
		rt[i] = bt[i];
	}

	printf("Enter Time Quantum: ");
	scanf("%d", &time_quantum);

	int remain = n;
	int time = 0;

	printf("\nProcess\t| Turnaround Time | Waiting Time\n\n");
	while (remain > 0)
	{
		for (int i = 0; i < n; i++)
		{
			if (rt[i] > 0)
			{
				if (rt[i] <= time_quantum)
				{
					time += rt[i];
					rt[i] = 0;
					tat[i] = time - at[i];
					wt[i] = tat[i] - bt[i];
					avg_wt += wt[i];
					avg_tat += tat[i];
					remain--;
					printf("P%d\t|\t%d\t|\t%d\n", i + 1, tat[i], wt[i]);
				}
				else
				{
					time += time_quantum;
					rt[i] -= time_quantum;
				}
			}
```

```
            }
        }

        avg_wt /= n;
        avg_tat /= n;
        printf("\nAverage Waiting Time: %.2f\n", avg_wt);
        printf("Average Turnaround Time: %.2f\n", avg_tat);

        return 0;
}
```

Output

```
Enter Total Number of Processes: 5
Enter Arrival Time and Burst Time for Each Process:
Process 1:
Arrival Time: 4
Burst Time: 5
Process 2:
Arrival Time: 2
Burst Time: 8
Process 3:
Arrival Time: 6
Burst Time: 4
Process 4:
Arrival Time: 1
Burst Time: 2
Process 5:
Arrival Time: 0
Burst Time: 7
Enter Time Quantum: 2

Process | Turnaround Time | Waiting Time

P4      |       7       |       5
P3      |       10      |       6
P1      |       15      |       10
P2      |       23      |       15
P5      |       26      |       19

Average Waiting Time: 11.00
Average Turnaround Time: 16.20
```

d) Priority

Code

```c
#include <stdio.h>

int main() {
    int bt[20], p[20], wt[20], tat[20], pr[20], n, total = 0, temp, avg_wt,
avg_tat;

    printf("Enter Total Number of Processes: ");
    scanf("%d", &n);

    printf("\nEnter Burst Time and Priority for Each Process:\n");
    for (int i = 0; i < n; i++) {
        printf("\nP[%d]\n", i + 1);
        printf("Burst Time: ");
        scanf("%d", &bt[i]);
        printf("Priority: ");
        scanf("%d", &pr[i]);
        p[i] = i + 1;
    }

    for (int i = 0; i < n; i++) {
        int pos = i;
        for (int j = i + 1; j < n; j++) {
            if (pr[j] < pr[pos])
                pos = j;
        }
        temp = pr[i]; pr[i] = pr[pos]; pr[pos] = temp;
        temp = bt[i]; bt[i] = bt[pos]; bt[pos] = temp;
        temp = p[i]; p[i] = p[pos]; p[pos] = temp;
    }

    wt[0] = 0;
    for (int i = 1; i < n; i++) {
        wt[i] = 0;
        for (int j = 0; j < i; j++)
            wt[i] += bt[j];
        total += wt[i];
    }

    avg_wt = total / n;
    total = 0;

    printf("\nProcess\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for (int i = 0; i < n; i++) {
        tat[i] = bt[i] + wt[i];
        total += tat[i];
        printf("P[%d]\t%d\t\t%d\t\t%d\n", p[i], bt[i], wt[i], tat[i]);
    }

    avg_tat = total / n;
    printf("\nAverage Waiting Time: %d", avg_wt);
    printf("\nAverage Turnaround Time: %d\n", avg_tat);

    return 0;
```

```
}
```

Output

```
Enter Total Number of Processes: 5

Enter Burst Time and Priority for Each Process:

P[1]
Burst Time: 4
Priority: 2

P[2]
Burst Time: 8
Priority: 1

P[3]
Burst Time: 9
Priority: 4

P[4]
Burst Time: 7
Priority: 3

P[5]
Burst Time: 7
Priority: 5

Process Burst Time      Waiting Time    Turnaround Time
P[2]    8               0               8
P[1]    4               8               12
P[4]    7               12              19
P[3]    9               19              28
P[5]    7               28              35

Average Waiting Time: 13
Average Turnaround Time: 20
```

3. Producer-Consumer Problem

Code

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>

#define BUFFER_SIZE 5
```

```c
#define MAX_ITEMS 20

int buffer[BUFFER_SIZE];
int in = 0, out = 0, produced_count = 0, consumed_count = 0;

sem_t mutex, full, empty;

void *producer(void *arg) {
    while (1) {
        sem_wait(&empty);
        sem_wait(&mutex);

        buffer[in] = produced_count + 1;
        printf("Produced: %d\n", buffer[in]);
        in = (in + 1) % BUFFER_SIZE;
        produced_count++;

        sem_post(&mutex);
        sem_post(&full);

        if (produced_count == MAX_ITEMS)
            break;
    }
    return NULL;
}

void *consumer(void *arg) {
    while (1) {
        sem_wait(&full);
        sem_wait(&mutex);

        int item = buffer[out];
        printf("Consumed: %d\n", item);
        out = (out + 1) % BUFFER_SIZE;
        consumed_count++;

        sem_post(&mutex);
        sem_post(&empty);

        if (consumed_count == MAX_ITEMS)
            break;
    }
    return NULL;
}

int main() {
    pthread_t producer_thread, consumer_thread;

    sem_init(&mutex, 0, 1);
    sem_init(&full, 0, 0);
    sem_init(&empty, 0, BUFFER_SIZE);

    pthread_create(&producer_thread, NULL, producer, NULL);
    pthread_create(&consumer_thread, NULL, consumer, NULL);
```

```
    pthread_join(producer_thread, NULL);
    pthread_join(consumer_thread, NULL);

    sem_destroy(&mutex);
    sem_destroy(&full);
    sem_destroy(&empty);

    return 0;
}
```

Output

```
Produced: 1
Produced: 2
Produced: 3
Consumed: 1
Consumed: 2
Consumed: 3
Produced: 4
Produced: 5
Produced: 6
Produced: 7
Produced: 8
Consumed: 4
Consumed: 5
Consumed: 6
Consumed: 7
Consumed: 8
Produced: 9
Produced: 10
Produced: 11
Produced: 12
Produced: 13
Consumed: 9
Consumed: 10
Consumed: 11
Consumed: 12
Consumed: 13
Produced: 14
Produced: 15
Consumed: 14
Consumed: 15
Produced: 16
Produced: 17
Produced: 18
Produced: 19
Produced: 20
Consumed: 16
```

```
Consumed: 17
Consumed: 18
Consumed: 19
Consumed: 20
```

4. Interprocess Communication

Code

Writer

```c
#include <stdio.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
      int fd;
      char buf[1024];

      /* create the FIFO (named pipe) */
      char *myfifo = "/tmp/myfifo";
      mkfifo(myfifo, 0666);
      printf("Run Reader process to read the FIFO File\n");

      fd = open(myfifo, O_WRONLY);
      write(fd, "Hi", sizeof("Hi"));
      /* write "Hi" to the FIFO */

      close(fd);
      unlink(myfifo); /* remove the FIFO */
      return 0;
}
```

Reader

```c
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#define MAX_BUF 1024
int main()
{
int fd;
```

```
/* A temp FIFO file is not created in reader */
char *myfifo = "/tmp/myfifo";
char buf[MAX_BUF];

/* open, read, and display the message from the FIFO */
fd = open(myfifo, O_RDONLY);
read(fd, buf, MAX_BUF);
printf("Writer: %s\n", buf);

close(fd);
return 0;
}
```

Output Writer

```
gcc writer.c -o writer
./writer
Run Reader process to read the FIFO File
```

Output Reader

```
gcc reader.c -o reader
./reader
Writer: Hi
```

5. Banker's Algorithm

Code

```
#include <stdio.h>
#define MAX_PROCESSES 10
#define MAX_RESOURCES 10

int main()
{
  int MAX[MAX_PROCESSES][MAX_RESOURCES], alloc[MAX_PROCESSES][MAX_RESOURCES],
avail[MAX_RESOURCES];
  int p, r, i, j, process, count = 0;
  int completed[MAX_PROCESSES] = {0}; // Array to track completed processes,
initialized to 0
  int safeSequence[MAX_PROCESSES] = {0};

  printf("Enter the number of processes: ");
  scanf("%d", &p);

  printf("Enter the number of resources: ");
```

```
  scanf("%d", &r);

  printf("\nEnter the max matrix for each process:\n");
  for (i = 0; i < p; i++)
  {
    printf("For process %d: ", i + 1);
    for (j = 0; j < r; j++)
      scanf("%d", &MAX[i][j]);
  }

  printf("\nEnter the allocation for each process:\n");
  for (i = 0; i < p; i++)
  {
    printf("For process %d: ", i + 1);
    for (j = 0; j < r; j++)
      scanf("%d", &alloc[i][j]);
  }

  printf("\nEnter the available resources:\n");
  for (i = 0; i < r; i++)
    scanf("%d", &avail[i]);

  do
  {
    process = -1; // Initialize process to indicate no process is selected
initially
    // Iterate over each process to find one that can be executed
    for (i = 0; i < p; i++)
    {
      if (!completed[i])
      {              // If the process is not completed
        process = i; // Mark the current process as a candidate for execution
        // Check if the available resources are sufficient to satisfy the needs of
this process
        for (j = 0; j < r; j++)
        {
          if (avail[j] < MAX[i][j] - alloc[i][j])
          {               // If not enough resources available
            process = -1; // Reset the process selection
            break;        // Exit the loop for this process
          }
        }
      }
      if (process != -1) // If a process has been selected for execution
        break;           // Exit the loop for process selection
    }
    if (process != -1)
    {                                                          // If a process has
been selected for execution
      printf("\nProcess %d runs to completion!", process + 1); // Print the
process number
      safeSequence[count] = i;
      // Release the allocated resources of the completed process
      for (j = 0; j < r; j++)
```

```
      {
        avail[j] += alloc[process][j]; // Return allocated resources to available
pool
        alloc[process][j] = 0;         // Reset the allocation for this process
        MAX[process][j] = 0;           // Reset the maximum requirement for this
process (optional)
        completed[process] = 1;        // Mark the process as completed
      }
      count++;
    }
  } while (count != p && process != -1); // Continue until all processes are
completed or deadlock is detected

  if (count == p)
  {
    printf("\nThe system is in a safe state!\n");
    printf("Safe Sequence: <");
        for (i = 0; i < p - 1; i++)
            printf("%d, ", safeSequence[i] +1);
        printf("%d>\n", safeSequence[p - 1]+ 1);

  }
  else
  {
    printf("\nThe system is in an unsafe state!\n");
  }

  return 0;
}
```

Output

```
Enter the number of processes: 5
Enter the number of resources: 3

Enter the max matrix for each process:
For process 1: 7 5 3
For process 2: 3 2 2
For process 3: 9 0 2
For process 4: 2 2 2
For process 5: 4 3 3

Enter the allocation for each process:
For process 1: 0 1 0
For process 2: 2 0 0
For process 3: 3 0 2
For process 4: 2 1 1
For process 5: 0 0 2

Enter the available resources:
```

```
3 3 2

Process 2 runs to completion!
Process 4 runs to completion!
Process 1 runs to completion!
Process 3 runs to completion!
Process 5 runs to completion!
The system is in a safe state!
Safe Sequence: <2, 4, 1, 3, 5>
```

6. Contiguous Memory Allocation

a) Worst Fit

Code

```
#include <stdio.h>

void worstFit(int blocksize[], int m, int processSize[], int n) {
    int allocation[n];
    for (int i = 0; i < n; i++) {
        allocation[i] = -1;
        int wstidx = -1;
        for (int j = 0; j < m; j++) {
            if (blocksize[j] >= processSize[i] && (wstidx == -1 ||
blocksize[wstidx] < blocksize[j])) {
                wstidx = j;
            }
        }
        if (wstidx != -1) {
            allocation[i] = wstidx;
            blocksize[wstidx] -= processSize[i];
        }
    }
    printf("Process No\tProcess Size\tBlock No.\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t\t%d\t\t", i + 1, processSize[i]);
        if (allocation[i] != -1)
            printf("%d", allocation[i] + 1);
        else
            printf("Not Allocated");
        printf("\n");
    }
}

int main() {
    int blocksize[] = {100, 500, 200, 300, 600};
    int processSize[] = {212, 417, 112, 426};
    int m = sizeof(blocksize) / sizeof(blocksize[0]);
    int n = sizeof(processSize) / sizeof(processSize[0]);
```

```
    worstFit(blocksize, m, processSize, n);
    return 0;
}
```

Output

```
Process No      Process Size    Block No.
1               212             5
2               417             2
3               112             5
4               426             Not Allocated
```

b) Best Fit

Code

```
#include <stdio.h>

void bestFit(int bs[], int m, int ps[], int n) {
    int allocation[n];
    for (int i = 0; i < n; i++) {
        allocation[i] = -1;
        for (int j = 0; j < m; j++) {
            if (bs[j] >= ps[i] && (allocation[i] == -1 || bs[allocation[i]] >
bs[j])) {
                allocation[i] = j;
            }
        }
        if (allocation[i] != -1) {
            bs[allocation[i]] -= ps[i];
        }
    }
    printf("\nProcess\tSize\tBlock\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t", i + 1, ps[i]);
        if (allocation[i] != -1)
            printf("%d", allocation[i] + 1);
        else
            printf("NA");
        printf("\n");
    }
}

int main() {
    int bs[] = {100, 50, 25, 430, 600};
    int ps[] = {20, 45, 100, 426};
    bestFit(bs, sizeof(bs) / sizeof(bs[0]), ps, sizeof(ps) / sizeof(ps[0]));
```

```
    return 0;
}
```

Output

```
Process Size    Block
1       20      3
2       45      2
3       100     1
4       426     4
```

c) First Fit

Code

```
#include <stdio.h>

void firstFit(int blockSizes[], int m, int processSizes[], int n) {
    int allocation[n];
    for (int i = 0; i < n; i++) {
        allocation[i] = -1;
        for (int j = 0; j < m; j++) {
            if (blockSizes[j] >= processSizes[i]) {
                allocation[i] = j;
                blockSizes[j] -= processSizes[i];
                break;
            }
        }
    }
    printf("Process No\tProcess Size\tBlock No\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t\t%d\t\t", i + 1, processSizes[i]);
        if (allocation[i] != -1)
            printf("%d", allocation[i] + 1);
        else
            printf("Not Allocated");
        printf("\n");
    }
}

int main() {
    int blockSizes[] = {100, 500, 200, 300, 50};
    int processSizes[] = {212, 190, 112, 250, 400};
    int m = sizeof(blockSizes) / sizeof(blockSizes[0]);
    int n = sizeof(processSizes) / sizeof(processSizes[0]);

    firstFit(blockSizes, m, processSizes, n);
```

```
    return 0;
}
```

Output

```
Process No      Process Size    Block No
1               212             2
2               190             2
3               112             3
4               250             4
5               400             Not Allocated
```

7. Page Replacement Algorithms

a) FIFO

Code

```
#include <stdio.h>

#define MAX_FRAMES 10
#define MAX_PAGES 50

int main() {
    int reference_string[MAX_PAGES], frames[MAX_FRAMES];
    int n, num_frames, page_faults = 0, frame_index = 0;

    // Input the number of pages
    printf("Enter the number of pages: ");
    scanf("%d", &n);

    // Input the reference string
    printf("Enter the reference string: ");
    for (int i = 0; i < n; i++)
        scanf("%d", &reference_string[i]);

    // Input the number of frames
    printf("Enter the number of frames: ");
    scanf("%d", &num_frames);
    for (int i = 0; i < MAX_FRAMES; i++)
        frames[i] = -1;
    // Iterate over the reference string
    for (int i = 0; i < n; i++) {
        int page_found = 0;

        // Check if page is already in frames
        for (int j = 0; j < num_frames; j++) {
```

```
            if (frames[j] == reference_string[i]) {
                page_found = 1;
                break;
            }
        }

        // If page is not in frames, replace the oldest page
        if (!page_found) {
            frames[frame_index] = reference_string[i];
            frame_index = (frame_index + 1) % num_frames;
            page_faults++;
        }

        // Print the frames
        for (int j = 0; j < num_frames; j++)
            printf("\t%d", frames[j]);
        printf("\n");
    }

    // Output the number of page faults
    printf("\nThe number of page faults is %d\n", page_faults);
    return 0;
}
```

Output

```
Enter the number of pages: 20
Enter the reference string: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
Enter the number of frames: 3
        7       -1      -1
        7       0       -1
        7       0       1
        2       0       1
        2       0       1
        2       3       1
        2       3       0
        4       3       0
        4       2       0
        4       2       3
        0       2       3
        0       2       3
        0       2       3
        0       1       3
        0       1       2
        0       1       2
        0       1       2
        7       1       2
        7       0       2
        7       0       1
```

```
The number of page faults is 15
```

b) LRU

Code

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

#define MAX_FRAMES 10

int main() {
    int reference_string[50], frames[MAX_FRAMES], n, num_frames, page_faults = 0,
frame_index = 0;
    int recent_counter[MAX_FRAMES] = {0};

    // Input the number of pages
    printf("Enter the number of pages: ");
    scanf("%d", &n);

    // Input the reference string
    printf("Enter the reference string: ");
    for (int i = 0; i < n; i++)
        scanf("%d", &reference_string[i]);

    // Input the number of frames
    printf("Enter the number of frames: ");
    scanf("%d", &num_frames);

    // Initialize frames array to -1 (indicating empty frame)
    for (int i = 0; i < num_frames; i++)
        frames[i] = -1;

    // Iterate over the reference string
    for (int i = 0; i < n; i++) {
        int page_found = 0;

        // Check if page is already in frames
        for (int j = 0; j < num_frames; j++) {
            if (frames[j] == reference_string[i]) {
                page_found = 1;
                recent_counter[j] = i;
                break;
            }
        }

        // If page is not in frames, replace a page with least recent access
        if (!page_found) {
            int lru_frame_index = 0;
            for (int j = 1; j < num_frames; j++) {
```

```
                if (recent_counter[j] < recent_counter[lru_frame_index])
                    lru_frame_index = j;
            }
            frames[lru_frame_index] = reference_string[i];
            recent_counter[lru_frame_index] = i;
            page_faults++;
        }

        // Print the frames
        for (int j = 0; j < num_frames; j++)
            printf("\t%d", frames[j]);
        printf("\n");
    }

    // Output the number of page faults
    printf("\nThe number of page faults is %d\n", page_faults);
    return 0;
}
```

Output

```
Enter the number of pages: 20
Enter the reference string: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
Enter the number of frames: 3
        7       -1      -1
        0       -1      -1
        0       1       -1
        0       1       2
        0       1       2
        0       3       2
        0       3       2
        0       3       4
        0       2       4
        3       2       4
        3       2       0
        3       2       0
        3       2       0
        3       2       1
        3       2       1
        0       2       1
        0       2       1
        0       7       1
        0       7       1
        0       7       1

The number of page faults is 12
```

8. File Organization Techniques

a) Single Level Directory

Code

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

#define MAX_FILES 10
#define MAX_FILENAME_LENGTH 10
#define MAX_DIRECTORY_NAME_LENGTH 10

struct Directory {
    char dname[MAX_DIRECTORY_NAME_LENGTH];
    char fname[MAX_FILES][MAX_FILENAME_LENGTH];
    int fcnt;
};

int main() {
    struct Directory dir;
    int ch, i;
    char fname[MAX_FILENAME_LENGTH];

    dir.fcnt = 0;

    printf("Enter name of directory: ");
    scanf("%s", dir.dname);

    while (1) {
        printf("\n\n1. Create File\t2. Delete File\t3. Search File\n4. Display
Files\t5. Exit\nEnter your choice: ");
        scanf("%d", &ch);

        switch (ch) {
            case 1:
                printf("Enter the name of the file: ");
                scanf("%s", dir.fname[dir.fcnt]);
                dir.fcnt++;
                break;
            case 2:
                printf("Enter the name of the file: ");
                scanf("%s", fname);
                for (i = 0; i < dir.fcnt; i++) {
                    if (strcmp(fname, dir.fname[i]) == 0) {
                        printf("File %s is deleted\n", fname);
                        strcpy(dir.fname[i], dir.fname[dir.fcnt - 1]);
                        dir.fcnt--;
                        break;
                    }
                }
                if (i == dir.fcnt)
                    printf("File %s not found\n", fname);
```

```
                break;
            case 3:
                printf("Enter the name of the file: ");
                scanf("%s", fname);
                for (i = 0; i < dir.fcnt; i++) {
                    if (strcmp(fname, dir.fname[i]) == 0) {
                        printf("File %s is found\n", fname);
                        break;
                    }
                }
                if (i == dir.fcnt)
                    printf("File %s not found\n", fname);
                break;
            case 4:
                if (dir.fcnt == 0)
                    printf("Directory is empty\n");
                else {
                    printf("Files in the directory: ");
                    for (i = 0; i < dir.fcnt; i++)
                        printf("\t%s", dir.fname[i]);
                    printf("\n");
                }
                break;
            case 5:
                exit(0);
            default:
                printf("Invalid choice\n");
                break;
        }
    }

    return 0;
}
```

Output

```
Enter name of directory: Documents


1. Create File  2. Delete File   3. Search File
4. Display Files    5. Exit
Enter your choice: 1
Enter the name of the file: file1


1. Create File  2. Delete File   3. Search File
4. Display Files    5. Exit
Enter your choice: 1
Enter the name of the file: file2
```

```
1. Create File  2. Delete File   3. Search File
4. Display Files    5. Exit
Enter your choice: 3
Enter the name of the file: file1
File file1 is found


1. Create File  2. Delete File   3. Search File
4. Display Files    5. Exit
Enter your choice: 3
Enter the name of the file: file3
File file3 not found


1. Create File  2. Delete File   3. Search File
4. Display Files    5. Exit
Enter your choice: 2
Enter the name of the file: file2
File file2 is deleted


1. Create File  2. Delete File   3. Search File
4. Display Files    5. Exit
Enter your choice: 4
Files in the directory:    file1


1. Create File  2. Delete File   3. Search File
4. Display Files    5. Exit
Enter your choice: 5
```

b) Two Level Directory

Code

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

#define MAX_DIRECTORIES 10
#define MAX_FILES_PER_DIRECTORY 10
#define MAX_NAME_LENGTH 30

struct Directory {
    char dname[MAX_NAME_LENGTH];
    char fname[MAX_FILES_PER_DIRECTORY][MAX_NAME_LENGTH];
    int fcnt;
};
```

```
int main() {
    struct Directory dir[MAX_DIRECTORIES];
    int ch, dcnt = 0, i, k;
    char d[MAX_NAME_LENGTH], f[MAX_NAME_LENGTH];

    while (1) {
        printf("\n\n1. Create Directory\t2. Create File\t3. Delete File\n4. Search
File\t\t5. Display\t6. Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &ch);

        switch (ch) {
            case 1:
                printf("\nEnter name of directory: ");
                scanf("%s", dir[dcnt].dname);
                dir[dcnt].fcnt = 0;
                dcnt++;
                printf("Directory created\n");
                break;
            case 2:
                printf("\nEnter name of the directory: ");
                scanf("%s", d);
                for (i = 0; i < dcnt; i++) {
                    if (strcmp(d, dir[i].dname) == 0) {
                        printf("Enter name of the file: ");
                        scanf("%s", dir[i].fname[dir[i].fcnt]);
                        dir[i].fcnt++;
                        printf("File created\n");
                        break;
                    }
                }
                if (i == dcnt)
                    printf("Directory %s not found\n", d);
                break;
            case 3:
                printf("\nEnter name of the directory: ");
                scanf("%s", d);
                for (i = 0; i < dcnt; i++) {
                    if (strcmp(d, dir[i].dname) == 0) {
                        printf("Enter name of the file: ");
                        scanf("%s", f);
                        for (k = 0; k < dir[i].fcnt; k++) {
                            if (strcmp(f, dir[i].fname[k]) == 0) {
                                printf("File %s is deleted\n", f);
                                dir[i].fcnt--;
                                strcpy(dir[i].fname[k],
dir[i].fname[dir[i].fcnt]);
                                break;
                            }
                        }
                        if (k == dir[i].fcnt)
                            printf("File %s not found\n", f);
                        break;
                    }
```

```
                }
                if (i == dcnt)
                    printf("Directory %s not found\n", d);
                break;
            case 4:
                printf("\nEnter name of the directory: ");
                scanf("%s", d);
                for (i = 0; i < dcnt; i++) {
                    if (strcmp(d, dir[i].dname) == 0) {
                        printf("Enter the name of the file: ");
                        scanf("%s", f);
                        for (k = 0; k < dir[i].fcnt; k++) {
                            if (strcmp(f, dir[i].fname[k]) == 0) {
                                printf("File %s is found\n", f);
                                break;
                            }
                        }
                        if (k == dir[i].fcnt)
                            printf("File %s not found\n", f);
                        break;
                    }
                }
                if (i == dcnt)
                    printf("Directory %s not found\n", d);
                break;
            case 5:
                if (dcnt == 0)
                    printf("\nNo Directories\n");
                else {
                    printf("\nDirectory\tFiles\n");
                    for (i = 0; i < dcnt; i++) {
                        printf("%s\t\t", dir[i].dname);
                        for (k = 0; k < dir[i].fcnt; k++)
                            printf("%s\t", dir[i].fname[k]);
                        printf("\n");
                    }
                }
                break;
            case 6:
                exit(0);
            default:
                printf("Invalid choice\n");
                break;
        }
    }

    return 0;
}
```

Output

```
1. Create Directory    2. Create File    3. Delete File
4. Search File         5. Display        6. Exit
Enter your choice: 1
Enter name of directory: Documents
Directory created

1. Create Directory    2. Create File    3. Delete File
4. Search File         5. Display        6. Exit
Enter your choice: 1
Enter name of directory: Pictures
Directory created

1. Create Directory    2. Create File    3. Delete File
4. Search File         5. Display        6. Exit
Enter your choice: 2
Enter name of the directory: Documents
Enter name of the file: doc1.txt
File created

1. Create Directory    2. Create File    3. Delete File
4. Search File         5. Display        6. Exit
Enter your choice: 2
Enter name of the directory: Pictures
Enter name of the file: pic1.jpg
File created

1. Create Directory    2. Create File    3. Delete File
4. Search File         5. Display        6. Exit
Enter your choice: 4
Enter name of the directory: Documents
Enter the name of the file: doc1.txt
File doc1.txt is found

1. Create Directory    2. Create File    3. Delete File
4. Search File         5. Display        6. Exit
Enter your choice: 4
Enter name of the directory: Documents
Enter the name of the file: doc2.txt
File doc2.txt not found

1. Create Directory    2. Create File    3. Delete File
4. Search File         5. Display        6. Exit
Enter your choice: 3
Enter name of the directory: Documents
Enter name of the file: doc1.txt
File doc1.txt is deleted

1. Create Directory    2. Create File    3. Delete File
4. Search File         5. Display        6. Exit
Enter your choice: 5
Directory     Files
Documents
Pictures      pic1.jpg
```

```
1. Create Directory    2. Create File    3. Delete File
4. Search File         5. Display        6. Exit
Enter your choice: 6
```

9. Linked File Allocation

Code

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

int main()
{
  int allocated[50] = {0}; // Initialize all blocks as unallocated
  int num_allocated, start, length, additional_blocks, choice;

  printf("Enter how many blocks are already allocated: ");
  scanf("%d", &num_allocated);

  printf("Enter the blocks already allocated: ");
  for (int i = 0; i < num_allocated; i++)
  {
    int block;
    scanf("%d", &block);
    allocated[block] = 1; // Mark the block as allocated
  }

  do
  {
    printf("Enter starting block index and length: ");
    scanf("%d %d", &start, &length);

    additional_blocks = 0;

    if (allocated[start] == 0)
    {
      for (int j = start; j < start + length; j++)
      {
        if (allocated[j] == 0)
        {
          allocated[j] = 1;
          printf("%d --------> %d\n", j, allocated[j]);
        }
        else
        {
          printf("%d Block is already allocated\n", j);
          length++; // Increment length if block is already allocated
          additional_blocks++;
        }
```

```
      }
    }
    else
    {
      printf("%d starting block is already allocated\n", start);
    }

    printf("Do you want to enter more file? (Yes - 1 / No - 0): ");
    scanf("%d", &choice);
  } while (choice == 1);

  return 0;
}
```

Output

```
Enter how many blocks are already allocated: 3
Enter the blocks already allocated: 1 3 5
Enter starting block index and length: 2 2
2 --------> 1
3 Block is already allocated
4 --------> 1
```

10. SCAN Disk Scheduling Algorithm

Code

```
#include <stdio.h>
int request[50];
int SIZE;
int pre;
int head;
int uptrack;
int downtrack;
struct max
{
    int up;
    int down;
} kate[50];
int dist(int a, int b)
{
    if (a > b)
        return a - b;
    return b - a;
}
void sort(int n)
{
    int i, j;
```

```
    for (i = 0; i < n - 1; i++)
    {
        for (j = 0; j < n - i - 1; j++)
        {
            if (request[j] > request[j + 1])
            {
                int temp = request[j];
                request[j] = request[j + 1];
                request[j + 1] = temp;
            }
        }
    }
    j = 0;
    i = 0;
    while (request[i] != head)
    {
        kate[j].down = request[i];
        j++;
        i++;
    }
    downtrack = j;
    i++;
    j = 0;
    while (i < n)
    {
        kate[j].up = request[i];
        j++;
        i++;
    }
    uptrack = j;
}
void scan(int n)
{
    int i;
    int seekcount = 0;
    printf("SEEK SEQUENCE = ");
    sort(n);
    if (pre < head)
    {
        for (i = 0; i < uptrack; i++)
        {
            printf("%d ", head);
            seekcount = seekcount + dist(head, kate[i].up);
            head = kate[i].up;
        }
        for (i = downtrack - 1; i > 0; i--)
        {
            printf("%d ", head);
            seekcount = seekcount + dist(head, kate[i].down);
            head = kate[i].down;
        }
    }
    else
    {
```

```
        for (i = downtrack - 1; i >= 0; i--)
        {
            printf("%d ", head);
            seekcount = seekcount + dist(head, kate[i].down);
            head = kate[i].down;
        }
        for (i = 0; i < uptrack - 1; i++)
        {
            printf("%d ", head);
            seekcount = seekcount + dist(head, kate[i].up);
            head = kate[i].up;
        }
    }
    printf(" %d\nTOTAL DISTANCE :%d", head, seekcount);
}
int main()
{
    int n, i;
    printf("ENTER THE DISK SIZE :\n");
    scanf("%d", &SIZE);
    printf("ENTER THE NO OF REQUEST SEQUENCE :\n");
    scanf("%d", &n);
    printf("ENTER THE REQUEST SEQUENCE :\n");
    for (i = 0; i < n; i++)
        scanf("%d", &request[i]);
    printf("ENTER THE CURRENT HEAD :\n");
    scanf("%d", &head);
    request[n] = head;
    request[n + 1] = SIZE - 1;
    request[n + 2] = 0;
    printf("ENTER THE PRE REQUEST :\n");
    scanf("%d", &pre);
    scan(n + 3);
}
```

Output

```
ENTER THE DISK SIZE :
4
ENTER THE NO OF REQUEST SEQUENCE :
2
ENTER THE REQUEST SEQUENCE :
1 2
ENTER THE CURRENT HEAD :
1
ENTER THE PRE REQUEST :
2
SEEK SEQUENCE = 1 0 1  2
TOTAL DISTANCE :3
```