UCS1411 Operating Systems Lab
AY: 2021-22

Name: Krithika Swaminathan
Roll No.: 205001057

Assignment 9 – Implementation of Paging Technique

Date: 23/05/2022

Aim:

To develop a C program to implement the paging technique in memory management.

Algorithm:

1. Start
2. Define structures for frames and processes. Initialize the frames with a value that indicates allocated but unknown processes, say, -1.
3. Get the total size of the physical memory and the size of each page as input from the user.
4. Divide the physical memory into frames and compute the number of free frames available.
5. Display a menu with operations for allocation and deallocation of processes, displaying the page table and displaying the free frames available.
    1. Allocating a process:
        1. Read the entering process ID and size as input from the user.
        2. Compute the number of pages required for allocating this process.
        3. If the number of free frames is sufficient, iterate through all the frames to identify the free ones and assign pages to them accordingly.
        4. If the number of free frames is not sufficient, print a suitable error message.
    2. Deallocating a process:
        1. Read the required process ID as input from the user.
        2. If all the frames are free or the required process is not found, alert the user to the same.
        3. If not, iterate through all the frames to find the one with the required process.
        4. Append the released frames to the free frame queue and mark them as unallocated.
    3. Displaying the page table:
        1. For each process, iterate through the allocated frames and print all the page-frame pairs for the process.
    4. Displaying the free frames available:
        1. Print the queue containing all the free frames.
6. Stop

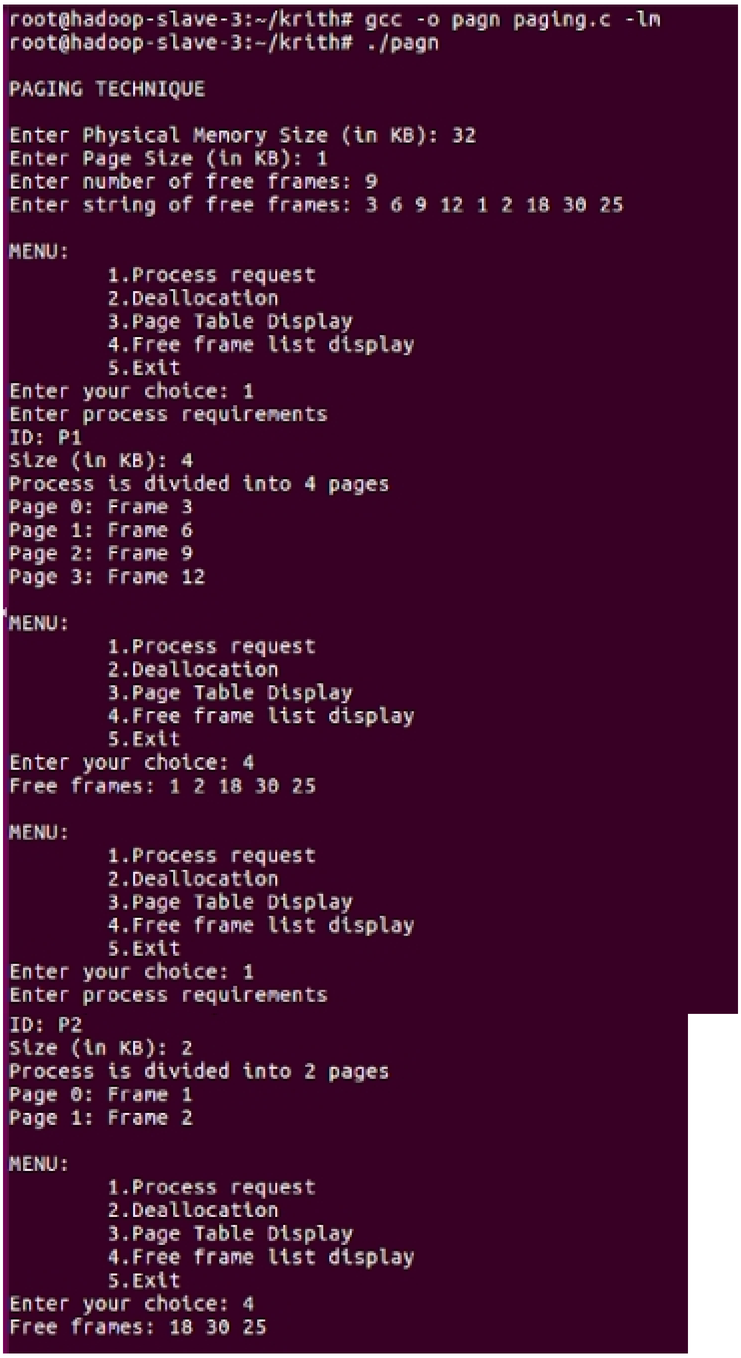
Program:

Code:

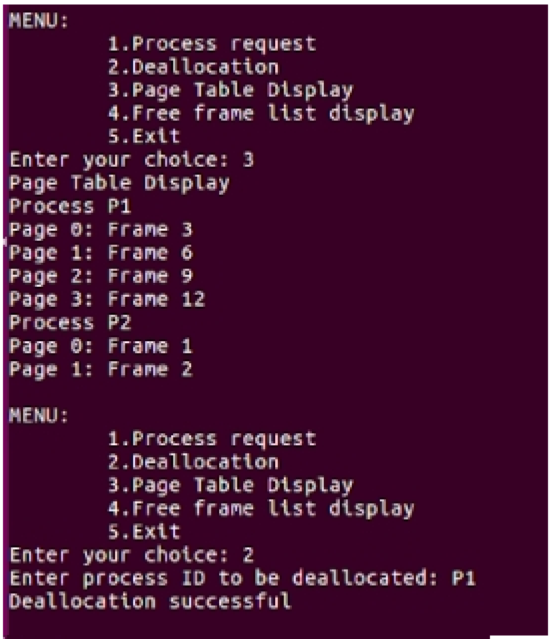
```
//Program to implement paging technique
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct frames {
  int allocated;
  int process;
  int page;
} frames;

typedef struct process {
  int pid;
  int npages;
  int pages[50];
} process;

void init(frames *F, int n) {
  for (int i = 0; i < n; i++) {
    F[i].allocated = 1;
    F[i].process = -1; // allocated but unknown
  }
}

int main() {
  int phymemsize, pagesize, n_frames;
  int pid, psize;
  int np = 0;
  printf("\nPAGING TECHNIQUE\n");
  printf("\nEnter Physical Memory Size (in KB): ");
  scanf("%d", &phymemsize);
  printf("Enter Page Size (in KB): ");
  scanf("%d", &pagesize);
  n_frames = ceil(phymemsize / pagesize);
  frames F[n_frames];
  init(F, n_frames);
  int n, inputfree[50];
  printf("Enter number of free frames: ");
  scanf("%d", &n);
  printf("Enter string of free frames: ");
  for (int i = 0; i < n; i++) {
    scanf("%d", &inputfree[i]);
    F[inputfree[i] - 1].allocated = 0;
    F[inputfree[i] - 1].process = -1; // free
```

```
}
int choice;
process P[50];
do {
  printf("\nMENU: \n\t1.Process request\n\t2.Deallocation\n\t3.Page "
         "Table Display\n\t4.Free frame list display\n\t5.Exit\n");
  printf("Enter your choice: ");
  scanf("%d", &choice);
  switch (choice) {
  case 1:
    // process request
    printf("Enter process requirements\nID: P");
    scanf("%d", &pid);
    printf("Size (in KB): ");
    scanf("%d", &psize);
    int nreqpages = ceil(psize / pagesize);
    if (n > nreqpages) {
      int temp = nreqpages;
      int tempfp[50];

      for (int i = 0; i < n; i++) {
        tempfp[i] = inputfree[i];
      }

      P[np].pid = pid;
      P[np].npages = temp;
      printf("Process is divided into %d pages\n", nreqpages);
      //
      // while (temp > 0)
      for (int i = 0; i < n && temp > 0; i++) {
        // if (F[tempfp[i]-1].allocated == 0)
        //{
        printf("Page %d: Frame %d\n", nreqpages - temp, tempfp[i]);
        P[np].pages[nreqpages - temp] = tempfp[i];
        F[tempfp[i] - 1].allocated = 1;
        F[tempfp[i] - 1].process = pid;
        F[tempfp[i] - 1].page = nreqpages - temp;
        temp--;
        n--;
        // updating process
        P[np].pages[nreqpages - temp] = tempfp[i];
        for (int j = 0; j < n; j++)
          inputfree[j] = inputfree[j + 1];
        //}
      }
      np++;
    } else {
      printf(
          "Unsuccessful. Insufficient free frames available for process.\n");
      }
      break;
    case 2:
      // deallocation
      if (n == n_frames)
        printf("All blocks are free frames.\n");
      else {
        printf("Enter process ID to be deallocated: P");
        scanf("%d", &pid);
        for (int i = 0; i < np; i++) {
          if (P[i].pid == pid) {
            for (int j = 0; j < P[i].npages; j++) {
              inputfree[n] = P[i].pages[j];
              n += 1;
            }
          }
        }
        np--;
        printf("Deallocation successful\n");
      }
      break;
    case 3:
      // pagetable display
      printf("Page Table Display\n");
      for (int i = 0; i < np; i++) {
        printf("Process P%d\n", P[i].pid);
        for (int j = 0; j < P[i].npages; j++) {
          printf("Page %d: Frame %d\n", j, P[i].pages[j]);
        }
      }
      break;
    case 4:
      // free frame display
      printf("Free frames: ");
      for (int i = 0; i < n; i++)
        printf("%d ", inputfree[i]);
      printf("\n");
      break;
    case 5:
      exit(0);
    default:
      printf("Invalid option entered.\n");
      break;
    }
  } while (choice != 5);
  return 0;
}
```

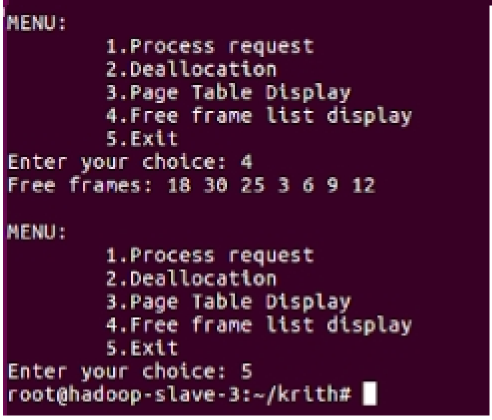
Output:



```
root@hadoop-slave-3:~/krith# gcc -o pagn paging.c -lm
root@hadoop-slave-3:~/krith# ./pagn

PAGING TECHNIQUE

Enter Physical Memory Size (in KB): 32
Enter Page Size (in KB): 1
Enter number of free frames: 9
Enter string of free frames: 3 6 9 12 1 2 18 30 25

MENU:
        1.Process request
        2.Deallocation
        3.Page Table Display
        4.Free frame list display
        5.Exit
Enter your choice: 1
Enter process requirements
ID: P1
Size (in KB): 4
Process is divided into 4 pages
Page 0: Frame 3
Page 1: Frame 6
Page 2: Frame 9
Page 3: Frame 12

MENU:
        1.Process request
        2.Deallocation
        3.Page Table Display
        4.Free frame list display
        5.Exit
Enter your choice: 4
Free frames: 1 2 18 30 25

MENU:
        1.Process request
        2.Deallocation
        3.Page Table Display
        4.Free frame list display
        5.Exit
Enter your choice: 1
Enter process requirements
ID: P2
Size (in KB): 2
Process is divided into 2 pages
Page 0: Frame 1
Page 1: Frame 2

MENU:
        1.Process request
        2.Deallocation
        3.Page Table Display
        4.Free frame list display
        5.Exit
Enter your choice: 4
Free frames: 18 30 25
```



```
MENU:
        1.Process request
        2.Deallocation
        3.Page Table Display
        4.Free frame list display
        5.Exit
Enter your choice: 3
Page Table Display
Process P1
Page 0: Frame 3
Page 1: Frame 6
Page 2: Frame 9
Page 3: Frame 12
Process P2
Page 0: Frame 1
Page 1: Frame 2

MENU:
        1.Process request
        2.Deallocation
        3.Page Table Display
        4.Free frame list display
        5.Exit
Enter your choice: 2
Enter process ID to be deallocated: P1
Deallocation successful
```



```
MENU:
        1.Process request
        2.Deallocation
        3.Page Table Display
        4.Free frame list display
        5.Exit
Enter your choice: 4
Free frames: 18 30 25 3 6 9 12

MENU:
        1.Process request
        2.Deallocation
        3.Page Table Display
        4.Free frame list display
        5.Exit
Enter your choice: 5
root@hadoop-slave-3:~/krith#
```

Learning outcomes:

- Paging in memory management was understood.
- Paging techniques were implemented.
- The concepts of page tables and frames were understood and implemented.