1. **WAP to demonstrate Paging**

**Ans.**

**AIM:** A program to demonstrate paging

**PROCEDURE:**

In Operating Systems, Paging is a storage mechanism used to retrieve processes from the secondary storage into the main memory in the form of pages. The main idea behind the paging is to divide each process in the form of pages. The main memory will also be divided in the form of frames. One page of the process is to be stored in one of the frames of the memory. The pages can be stored at the different locations of the memory but the priority is always to find the contiguous frames or holes. Pages of the process are brought into the main memory only when they are required otherwise they reside in the secondary storage. Different operating system defines different frame sizes. The sizes of each frame must be equal. Considering the fact that the pages are mapped to the frames in Paging, page size needs to be as same as frame size.

#include <stdio.h>

#define MAX\_PAGES 50

int main() {

int page[MAX\_PAGES], i, n, f, ps, off, pno;

int choice = 0;

printf("\nEnter the number of pages in memory (max %d): ", MAX\_PAGES);

scanf("%d", &n);

if (n > MAX\_PAGES || n <= 0) {

printf("Invalid number of pages.\n");

return 1;

}

printf("\nEnter page size: ");

scanf("%d", &ps);

if (ps <= 0) {

printf("Invalid page size.\n");

return 1;

}

printf("\nEnter number of frames: ");

scanf("%d", &f);

if (f <= 0 || f > n) {

printf("Invalid number of frames.\n");

return 1;

}

for (i = 0; i < n; i++)

page[i] = -1;

printf("\nEnter the page table\n");

printf("(Enter frame no as -1 if that page is not present in any frame)\n\n");

printf("\nPage No\tFrame No\n-------\t-------\n");

for (i = 0; i < n; i++) {

printf("%d\t\t", i);

scanf("%d", &page[i]);

}

do {

printf("\n\nEnter the logical address (page no & offset): ");

scanf("%d%d", &pno, &off);

if (pno < 0 || pno >= n) {

printf("Invalid page number.\n");

} else if (page[pno] == -1) {

printf("The required page is not available in any of frames.\n");

} else {

printf("Physical address (frame no & offset): %d, %d\n", page[pno], off);

}

printf("Do you want to continue (1/0)?: ");

scanf("%d", &choice);

} while (choice == 1);

return 0;

}

**A screenshot of a computer program

Description automatically generated**

**2.WAP to demonstrate segmentation**

**Ans.**

**AIM:** A program to demonstrate segmentation

**PROCEDURE:**

Segmentation is a memory management technique that splits up the virtual address space of an application into chunks. By splitting the memory up into manageable chunks, the operating system can track which parts of the memory are in use and which parts are free. This makes allocating and deallocating memory much faster and simpler for the operating system. The segments are of unequal size and are not placed in a contiguous way. As it’s a non-contiguous memory allocation technique, internal fragmentation doesn’t occur. The length is decided on the base of the purpose of the segment in a user program. Segmentation is a memory management technique which divides the program from the user’s view of memory. That means the program is divided into modules/segments, unlike paging in which the program was divided into different pages, and those pages may or may not be loaded into the memory simultaneously. Segmentation prevents internal fragmentation.

#include <stdio.h>

#include <stdio.h>

#define MAX\_SEGMENTS 10

#define MAX\_SEGMENT\_SIZE 10

#define MAX\_ELEMENTS (MAX\_SEGMENTS \* MAX\_SEGMENT\_SIZE)

int main()

{

int segments[MAX\_SEGMENTS][2]; // Array to store segment information: size and base address

int elements[MAX\_ELEMENTS]; // Array to store all elements

int i, j, n, seg, off, abs;

printf("Enter the number of segments (up to %d): ", MAX\_SEGMENTS);

scanf("%d", &n);

if (n <= 0 || n > MAX\_SEGMENTS)

{

printf("Invalid number of segments.\n");

return 1;

}

// Input segment information

for (i = 0; i < n; i++)

{

printf("Enter the size of segment %d: ", i );

scanf("%d", &segments[i][0]); // Size

printf("Enter the base address of segment %d: ", i );

scanf("%d", &segments[i][1]); // Base address

// Input elements for the segment

printf("Enter %d elements for segment %d:\n", segments[i][0], i );

for (j = 0; j < segments[i][0]; j++)

{

scanf("%d", &elements[segments[i][1] + j]); // Store elements at calculated address

}

}

// Input segment number and offset

printf("Enter the segment number and offset value: ");

scanf("%d %d", &seg, &off);

if (seg < 0 || seg >= n) {

printf("Invalid segment number.\n");

return 1;

}

if (off < 0 || off >= segments[seg][0]) {

printf("Error: Offset is out of bounds for segment %d.\n", seg);

return 1;

}

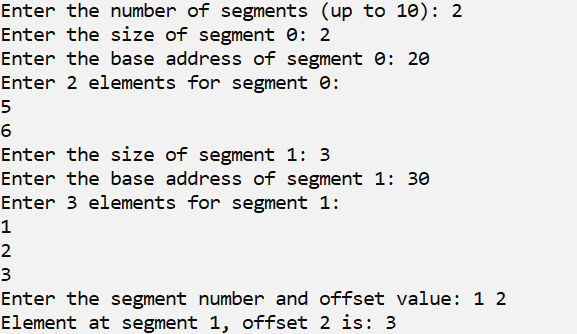
//calculate absolute address

abs = segments[seg][1] + off;

printf("Element at segment %d, offset %d is: %d\n", seg, off, elements[abs]);

return 0;

}

****

**4. WAP to demonstrate the use of signal – IPC (user defined Handler)**

**Ans.**

**AIM:** A program to demonstrate the use of signal - IPC

**DESCRIPTION:**

A signal is a software generated interrupt that is sent to a process by the OS because of when user press ctrl-c or another process tell something to this process. There are fix set of signals that can be sent to a process. signal are identified by integers. Signal number have symbolic names. For example SIGCHLD is number of the signal sent to the parent process when child terminates.

**CODE:**

#include<stdio.h>

#include<unistd.h>

#include<signal.h>

// Handler for SIGINT, caused by

// Ctrl-C at keyboard

void handle\_sigint(int sig)    // Signal Handler

{

    printf("Caught signal %d\n", sig);

}

int main()

{

    signal(SIGINT, handle\_sigint); // Registering a Signal

    while (1)

    {

        printf(“hello world\n”);

        sleep(1);

    }

    return 0;

}

**OUTPUT:**

A computer screen shot of a program code

Description automatically generated

**WAP to demonstrate the SIGCHLD and SIGINT handling**

**Ans.**

**AIM:** A program to demonstrate the SIGCHLD and SIGINT handling

**DESCRIPTION:**

A signal is a software generated interrupt that is sent to a process by the OS because of when user press ctrl-c or another process tell something to this process. There are fix set of signals that can be sent to a process. signal are identified by integers. Signal number have symbolic names. For example SIGCHLD is number of the signal sent to the parent process when child terminates.

**SIGCHLD handling:**

#include<stdio.h>

#include<signal.h>

#include<unistd.h>

// Handler for SIGINT, caused by

// Ctrl-C at keyboard

void handle\_sigchild(int sig) // Signal Handler

{

printf("Inside child signal %d \n", sig);

//signal(SIGINT,SIG\_DFL); //De registering Signal

}

int main()

{

signal(SIGCHLD, handle\_sigchild); // Registering a Signal

int p=fork();

if(p==0)

{

printf("Inside Child process \n");

//

printf("End of child Process \n");

}

else if(p>=0)

{

printf("Inside parent process \n");

wait();

printf("End of parent process \n");

}

return 0;

}

**OUTPUT:**

**SIGINT:**

A computer screen shot of a program code

Description automatically generated

**SIGCHLD:**

A screen shot of a computer program

Description automatically generated