**Task 1:** Implement the Paging algorithm in C language.

**Code:**

#include <stdio.h>

#define MAX\_PROCESSES 10

#define MAX\_PAGES 100

int main() {

int memoryLimit, pageSize, numOfProcesses;

int numOfPages, remainingPages;

int pageTable[MAX\_PROCESSES][MAX\_PAGES];

int pagesAllocated[MAX\_PROCESSES];

printf("Enter memory limit: ");

scanf("%d", &memoryLimit);

printf("Enter page size: ");

scanf("%d", &pageSize);

printf("Enter number of processes: ");

scanf("%d", &numOfProcesses);

numOfPages = memoryLimit / pageSize;

remainingPages = numOfPages;

printf("Number of pages available: %d\n", numOfPages);

for (int i = 0; i < numOfProcesses; i++) {

int pagesRequired;

printf("Enter the number of pages required by process %d: ", i + 1);

scanf("%d", &pagesRequired);

if (pagesRequired > remainingPages) {

printf("Not enough memory available for process %d.\n", i + 1);

continue;

}

remainingPages -= pagesRequired;

pagesAllocated[i] = pagesRequired;

printf("Enter the page table for process %d:\n", i + 1);

for (int j = 0; j < pagesRequired; j++) {

printf("Page %d: ", j);

scanf("%d", &pageTable[i][j]);}}

int processNumber, pageNumber, offset;

printf("Enter process number, page number, and offset to find physical address:\n");

printf("Process number: ");

scanf("%d", &processNumber);

printf("Page number: ");

scanf("%d", &pageNumber);

printf("Offset: ");

scanf("%d", &offset);

if (processNumber >= numOfProcesses || pageNumber >= pagesAllocated[processNumber] || offset >= pageSize) {

printf("Invalid entry.\n");

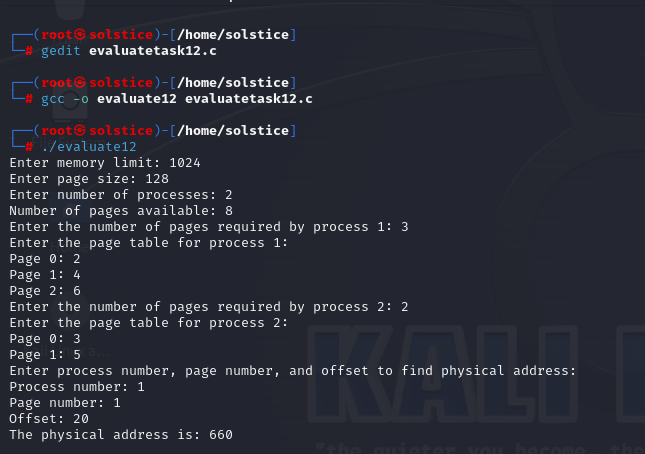
} else {

int physicalAddress = pageTable[processNumber][pageNumber] \* pageSize + offset;

printf("The physical address is: %d\n", physicalAddress); }

return 0;}

**Output:**



**Task 2:** Write a short note the Segmentation algorithm stating its main purpose and working mechanism.

**Main Purpose**:

The Segmentation algorithm in operating systems facilitates flexible memory allocation by dividing memory into variable-sized segments. Its primary goal is to accommodate the diverse memory requirements of processes efficiently.

**Working Mechanism:**

* Segmentation divides memory into segments, each representing a logical unit of a process (e.g., code, data).
* Each segment is assigned a unique segment number and has its base address and limit stored in a segmentation table.
* When a process accesses memory, it specifies the segment number and offset within the segment.
* The operating system translates this logical address into a physical address by adding the base address of the corresponding segment to the offset.
* Segmentation also provides memory protection by associating access rights with segments, controlling process access to memory.

**Task 3:** Implement the Segmentation algorithm, write the C language program for it.

**Code:**

#include <stdio.h>

#include <stdbool.h>

#define MAX\_SEGMENTS 10

typedef struct {

int base\_address;

int limit;

} Segment;

Segment segments[MAX\_SEGMENTS];

void create\_segment\_table(int num\_segments) {

for (int i = 0; i < num\_segments; i++) {

printf("Enter base address for segment %d: ", i + 1);

scanf("%d", &segments[i].base\_address);

printf("Enter limit for segment %d: ", i + 1);

scanf("%d", &segments[i].limit);

}

}

int main() {

int memory\_limit, num\_segments, segment\_number, offset;

printf("Enter memory limit: ");

scanf("%d", &memory\_limit);

printf("Enter number of segments: ");

scanf("%d", &num\_segments);

if (memory\_limit < 0) {

printf("Invalid memory limit\n");

return 0;

}

create\_segment\_table(num\_segments);

printf("Enter logical address (segment number offset): ");

scanf("%d %d", &segment\_number, &offset);

if (segment\_number < 0 || segment\_number >= num\_segments) {

printf("Invalid segment number\n");

return 0;

}

if (offset < segments[segment\_number].limit) {

int physical\_address = segments[segment\_number].base\_address + offset;

printf("Physical address: %d\n", physical\_address);

} else {

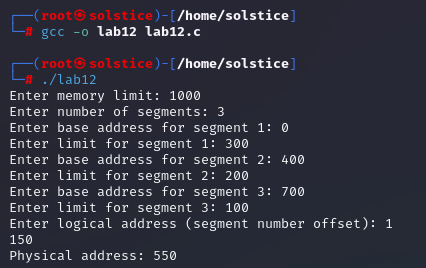
printf("Invalid offset\n");

}

return 0;

}

**Output:**

****