1. **/\* Left rotate k times \*/**
2. **#include <math.h>**
3. **#include <stdio.h>**
4. **#include <string.h>**
5. **#include <stdlib.h>**
6. **#include <assert.h>**
7. **#include <limits.h>**
8. **#include <stdbool.h>**
9. **int main(){**
10. **int n;**
11. **int k;**
12. **scanf("%d %d",&n,&k);**
13. **int \*a = malloc(sizeof(int) \* n);**
14. **for(int a\_i = 0; a\_i < n; a\_i++){**
15. **scanf("%d",&a[(a\_i+n-k)%n]);**
16. **}**
17. **for(int i = 0;i<n;i++)**
18. **printf("%d ",a[i]);**
19. **return 0;**
20. **}**
21. **/\* Left rotate k times \*/ //Alternative**
22. **#include <stdio.h>**
23. **int main()**
24. **{**
25. **int i,n,k;**
26. **scanf("%d%d",&n,&k);**
27. **int arr[2\*n];**
28. **for(i=0;i<n;i++)**
29. **{**
30. **scanf("%d",&arr[i]);**
31. **arr[n+i]=arr[i];**
32. **}**
33. **for(i=0;i<n;i++)**
34. **printf("%d ",arr[i+k]);**
35. **}**

**Memory Layout of C Programs**

A typical memory representation of a C program consists of the following sections.  
1. Text segment   
2. Initialized data segment   
3. Uninitialized data segment   
4. Stack   
5. Heap



A typical memory layout of a running process  
**1. Text Segment:**   
A text segment, also known as a code segment or simply as text, is one of the sections of a program in an object file or in memory, which contains executable instructions.  
As a memory region, a text segment may be placed below the heap or stack in order to prevent heaps and stack overflows from overwriting it. 

Usually, the text segment is sharable so that only a single copy needs to be in memory for frequently executed programs, such as text editors, the C compiler, the shells, and so on. Also, the text segment is often read-only, to prevent a program from accidentally modifying its instructions.

**2. Initialized Data Segment:**   
Initialized data segment, usually called simply the Data Segment. A data segment is a portion of the virtual address space of a program, which contains the global variables and static variables that are initialized by the programmer.  
Note that, the data segment is not read-only, since the values of the variables can be altered at run time.  
This segment can be further classified into the initialized read-only area and the initialized read-write area.  
For instance, the global string defined by char s[] = “hello world” in C and a C statement like int debug=1 outside the main (i.e. global) would be stored in the initialized read-write area. And a global C statement like const char\* string = “hello world” makes the string literal “hello world” to be stored in the initialized read-only area and the character pointer variable string in the initialized read-write area.  
Ex: static int i = 10 will be stored in the data segment and global int i = 10 will also be stored in data segment

**3. Uninitialized Data Segment:**   
Uninitialized data segment often called the “**bss**” segment, named after an ancient assembler operator that stood for “**block started by symbol**.” Data in this segment is initialized by the kernel to arithmetic 0 before the program starts executing  
uninitialized data starts at the end of the data segment and contains all global variables and static variables that are initialized to zero or do not have explicit initialization in source code.  
For instance, a variable declared static int i; would be contained in the BSS segment.   
For instance, a global variable declared int j; would be contained in the BSS segment.

**4. Stack:**   
The stack area traditionally adjoined the heap area and grew in the opposite direction; when the stack pointer met the heap pointer, free memory was exhausted. (With modern large address spaces and virtual memory techniques they may be placed almost anywhere, but they still typically grow in opposite directions.)  
The stack area contains the program stack, a LIFO structure, typically located in the higher parts of memory. On the standard PC x86 computer architecture, it grows toward address zero; on some other architectures, it grows in the opposite direction. A “stack pointer” register tracks the top of the stack; it is adjusted each time a value is “pushed” onto the stack. The set of values pushed for one function call is termed a “stack frame”; A stack frame consists at minimum of a return address.  
Stack, where automatic variables are stored, along with information that is saved each time a function is called. Each time a function is called, the address of where to return to and certain information about the caller’s environment, such as some of the machine registers, are saved on the stack. The newly called function then allocates room on the stack for its automatic and temporary variables. This is how recursive functions in C can work. Each time a recursive function calls itself, a new stack frame is used, so one set of variables doesn’t interfere with the variables from another instance of the function.

**5. Heap:**   
Heap is the segment where dynamic memory allocation usually takes place.  
The heap area begins at the end of the BSS segment and grows to larger addresses from there. The Heap area is managed by malloc, realloc, and free, which may use the brk and sbrk system calls to adjust its size (note that the use of brk/sbrk and a single “heap area” is not required to fulfill the contract of malloc/realloc/free; they may also be implemented using mmap to reserve potentially non-contiguous regions of virtual memory into the process’ virtual address space). The Heap area is shared by all shared libraries and dynamically loaded modules in a process.

Examples.  
The size(1) command reports the sizes (in bytes) of the text, data, and bss segments. ( for more details please refer man page of size(1) )

**Q. What is Sampling ?**

Recording an analog signal at evenly spaced instants in time creates samples.

Sampling is the process of recording an analog signal at regular discrete moments of time. The sampling rate fs is the number of samples per second.

The time interval between samples is called the sampling interval Ts=1/fs.

**Q. Number Of Elemnets in Array without “sizeof” operator.**

#include <stdio.h>

int main()

{

    int  arr[] = {1, 2, 3, 4, 5, 6};

    int size = \*(&arr + 1) - arr;

    printf("Number of elements in arr[] is %d",size);

    return 0;

}

**Q. Size of Array without “sizeof” operator.**

#include <stdio.h>

int main()

{

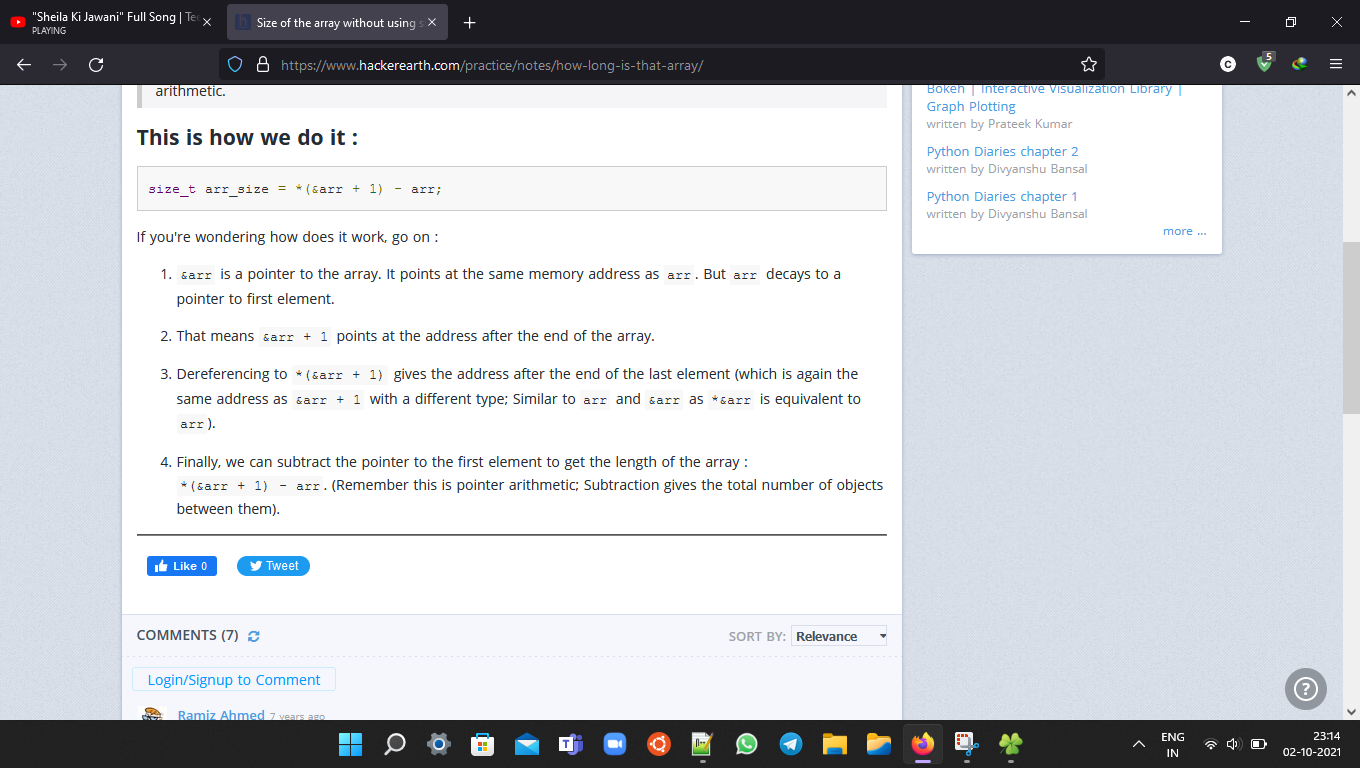
int arr[] = {1, 2, 3, 4, 5, 6};

int size = (char\*) \*(&arr + 1) - (char\*)arr;

printf("Size of arr[] is %d",size);

return 0;

}



# Q. Determine if two integers are equal without using comparison(==) and arithmetic operators(+,-,\*,%).

# 

# 

Q. Can a inline function be recursive?

**No,** An inline function cannot be recursive because in case of inline function the code is merely placed into the position from where it is called and does not maintain an information on **stack which is necessary for recursion.**

inlining is only a request to the compiler, not a command. Compiler can ignore the request for inlining. Compiler may not perform inlining in such circumstances like:  
1) If a function contains a loop. (for, while, do-while)  
2) If a function contains static variables.  
3) If a function is recursive.  
4) If a function return type is other than void, and the return statement doesn’t exist in function body.  
5) If a function contains switch or goto statement.

Q. can we implement strcpy() using macro?

#define \_strcpy(dst, src) \

do { \

const char \*\_src = (src); \

char \*\_dst = (dst); \

\

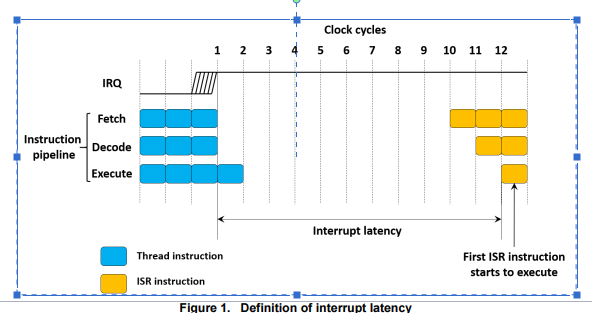
while ((\*\_dst++ = \*\_src++)) \

; \

} while (0)

**16.what is interrupt latency?how to reduce the time of interrupt ?**

The term interrupt latency refers to the delay between the start of an Interrupt Request (IRQ) and the start of the respective Interrupt Service Routine (ISR). The interrupt latency is expressed in core clock cycles. There is another exact definition-the number of clock cycles from the assertion of the interrupt request to the first ISR instruction executed, as shown in Figure 1.



7.which algorithm is best to sort a singly linked list with least time complexity?

[Merge sort](http://en.wikipedia.org/wiki/Merge_sort) is often preferred for sorting a linked list. The slow random-access performance of a linked list makes some other algorithms (such as quicksort) perform poorly, and others (such as heapsort) completely impossible.

Q. what happens when interrupt is rised?

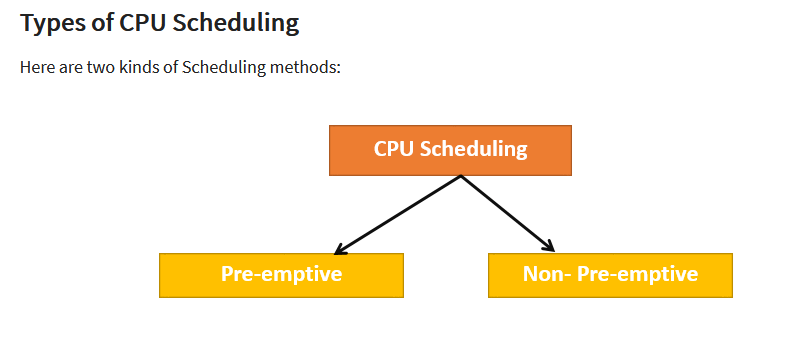
When a device raises an interrupt at let’s say process i, the processor first completes the execution of instruction i. Then it loads the Program Counter (PC) with the address of the first instruction of the ISR. Before loading the Program Counter with the address, the address of the interrupted instruction is moved to a temporary location. Therefore, after handling the interrupt the processor can continue with process i+1.

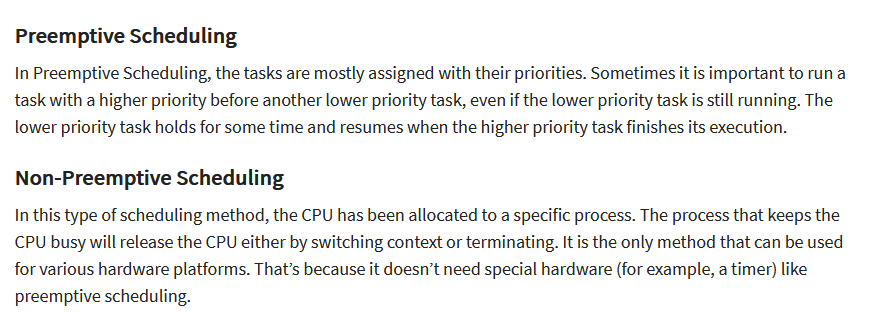
While the processor is handling the interrupts, it must inform the device that its request has been recognized so that it stops sending the interrupt request signal. Also, saving the registers so that the interrupted process can be restored in the future, increases the delay between the time an interrupt is received and the start of the execution of the ISR. This is called Interrupt Latency.

## **Q.What is CPU Scheduling?**

**CPU Scheduling** is a process of determining which process will own CPU for execution while another process is on hold. The main task of CPU scheduling is to make sure that whenever the CPU remains idle, the OS at least select one of the processes available in the ready queue for execution. The selection process will be carried out by the CPU scheduler. It selects one of the processes in memory that are ready for execution.

**Q.what are different scheduling algorithms?**





## 🡺 Types of CPU scheduling Algorithm

There are mainly six types of process scheduling algorithms

1. First Come First Serve (FCFS)
2. Shortest-Job-First (SJF) Scheduling
3. Shortest Remaining Time
4. Priority Scheduling
5. Round Robin Scheduling
6. Multilevel Queue Scheduling

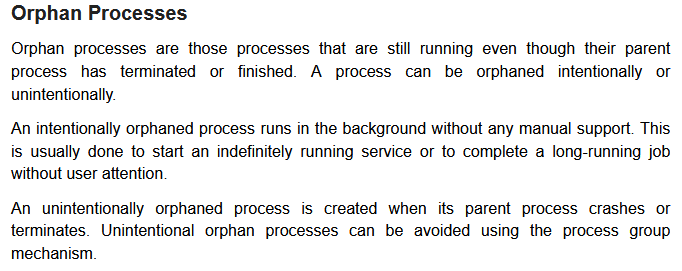
Q. what is race condition?

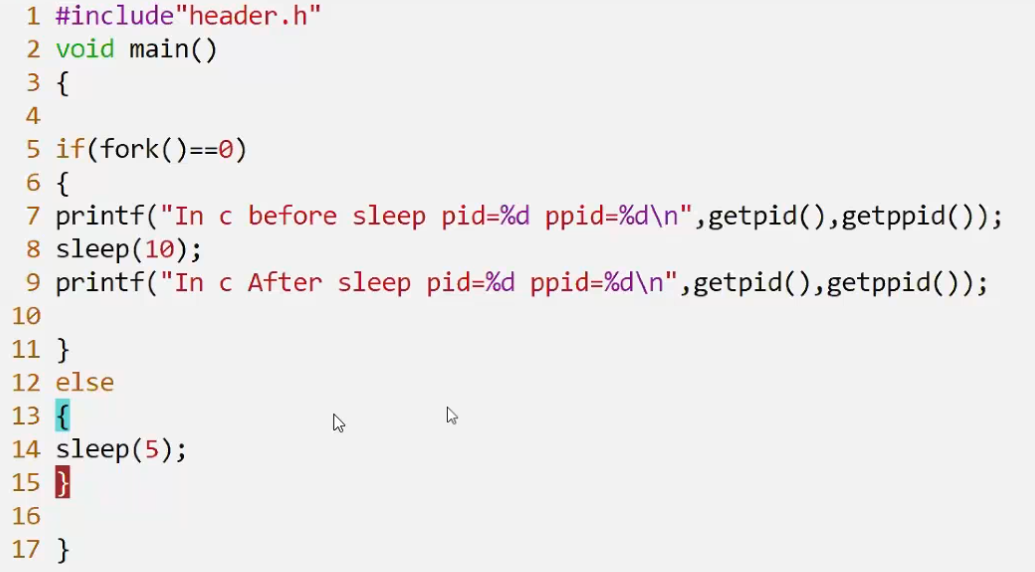
“ A race condition is an undesirable situation that occurs when a device or system attempts to perform two or more operations at the same time, but because of the nature of the device or system, the operations must be done in the proper sequence to be done correctly. ”

In simple words when multiple processes are in ready condition and waithing for time slice from CPU, which process get that time slice among them is called that process won that Race . And this process is called race condition.

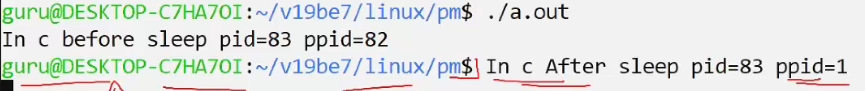
Q.what is orphan process?

if parent excution is completed first before child then the child become orphan. And for this orphan process init(1) becomes parent.





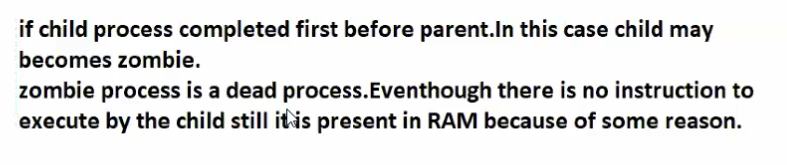
Output:



After ./a.out completing bash will be the parent for child process . Bash was waithing to complete ./a.out process as soon as ./a.out completed bash again print the cmd prompt to take input that is why cmd prompt is printed between before sleep(./a.out process output) and after sleep (after sleep is the child process output).

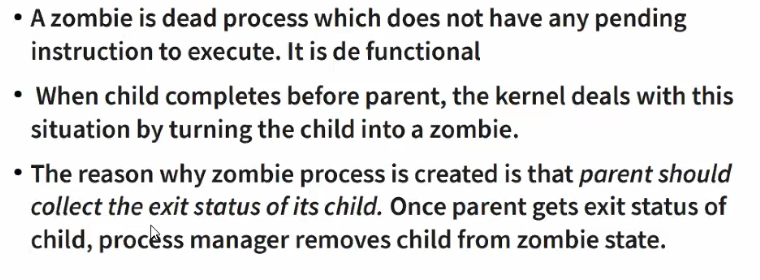
NOTE: [in this program ./a.out is completed first after that child process completed]

Q.What is Zombie Process?



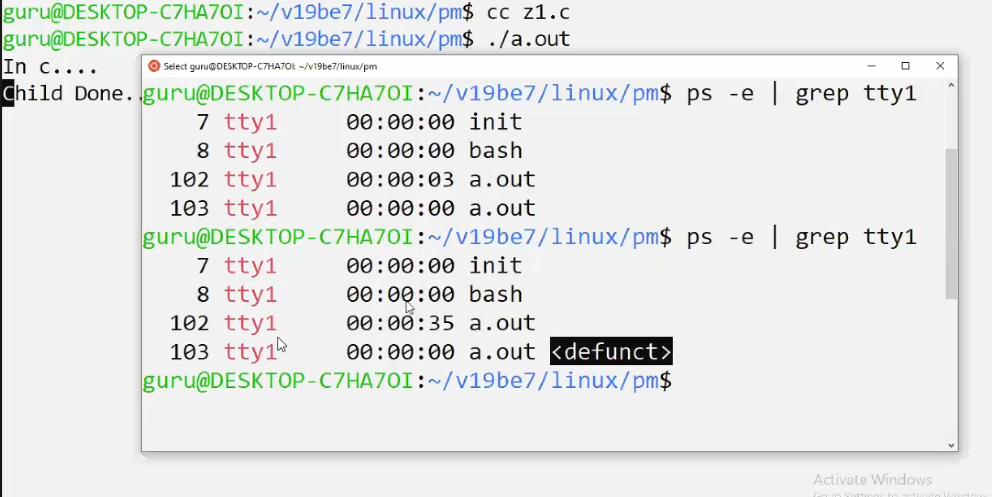
## **Zombie Processes**

A zombie process is a process whose execution is completed but it still has an entry in the process table. Zombie processes usually occur for child processes, as the parent process still needs to read its child’s exit status. Once this is done using the wait system call, the zombie process is eliminated from the process table. This is known as reaping the zombie process.

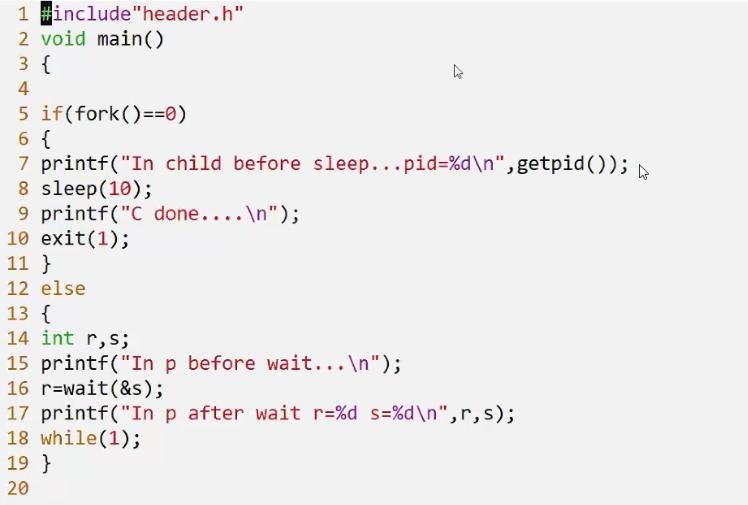


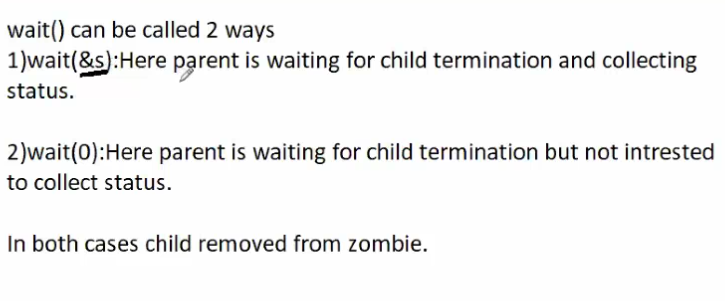


Output with explanation:



**EXIT(),\_EXIT(),WAIT() :**

****

****