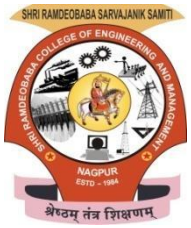


SHRI RAMDEOBABA COLLEGE OF ENGINEERING AND MANAGEMENT

Department of Computer Science & Engineering

Session: 2018-19

OPERATING SYSTEM

ASSIGNMENT

IV Semester, B.E.

Shift II

Group Members

Vishal Kriplani

Piyush Chotiya

Ateefa Ateeque

Course Co-ordinator

Prof.Heena Agrawal

Statement-19:A

Pizza-Problem (Using shared memory synchronization with busy-waiting)

Introduction

Three persons A, B, and C attend a game of housy whose prize is a jumbo pizza. The person whose coupon gets exhausted first gets first access to the pizza. The second person whose coupon gets exhausted gets second access and finally the third.

If multiple coupons of persons get exhausted simultaneously, then those persons can access the pizza in any arbitrary order between them.

Game begins

The three persons enter the room where the housy game is being held, with their respective

coupons. The judge J starts calling out numbers one by one. After a number is called out, the three persons up date their coupons by marking the number just called out. Only after the three persons have finished updating their coupons ,does the judge call out the next number. If all the numbers in the coupon for a person get marked, he goes ahead to have his share of pizza. The order of access to pizza must be strictly maintained.

Question

Write code for the processes A,B,C and J? The coupons are to be modelled as files containing lists of numbers. The numbers (between 1 to 100) are generated by J from a fixed test file , and the coupons contain 5 numbers each. There quired synchronization must be performed using shared variables[Linux Shared Memory] with atomic reads and writes .The solution must be devoid of unnecessary-delays, and starvation.

Don't Use semaphores. Only Shared Memory Must be Used.

//

Introduction

Shared Memory:

Inter Process Communication(IPC) through shared memory is a concept where two or more process can access the common memory and communication is done via this shared memory where changes made by one process can be viewed by anther process.

- Server reads from the input file.
- The server writes this data in a message using either a pipe, fifo or message queue.
- The client reads the data from the IPC channel, again requiring the data to be copied from kernel's IPC buffer to the client's buffer.
- Finally the data is copied from the client's buffer.

A total of four copies of data are required (2 read and 1 write). So, shared memory provides a way by letting two or more processes share a memory segment. With Shared Memory the data is only copied twice – from input file into shared memory and from shared memory to the output file.

Busy Waiting:

- The repeated execution of a loop of code while waiting for an event to occur is called busy-waiting. The CPU is not engaged in any real productive activity during this period, and the process does not progress toward completion.
- Busy waiting means a process simply spins, (does nothing but continue to test its entry condition) while it is waiting to enter its critical section. This continues to use (waste) CPU cycles, which is inefficient

Shmget():

#include <sys/ipc.h>

#include <sys/shm.h>

int shmget(key_t key, size_t size, int shmflg) The above system call creates or allocates a System V shared memory segment. The arguments that need to be passed are as follows −

The first argument, key, recognizes the shared memory segment. The key can be either an arbitrary value or one that can be derived from the library function ftok(). The key can also be IPC_PRIVATE, means, running processes as server and client (parent and child relationship) i.e., inter-related process communication. If the client wants to use shared memory with this key, then it must be a child process of the server. Also, the child process needs to be created after the parent has obtained a shared memory.

The second argument, size, is the size of the shared memory segment rounded to multiple of PAGE_SIZE. The third argument, shmflg, specifies the required shared memory flag/s such as IPC_CREAT (creating new segment) or IPC_EXCL (Used with IPC_CREAT to create new segment and the call fails, if the segment already exists). Need to pass the permissions as well.

Shmat():

#include <sys/types.h>

#include <sys/shm.h>

void * shmat(int shmid, consts void *shmaddr, int shmflg)

The above system call performs shared memory operation for System V shared memory segment i.e., attaching a shared memory segment to the address space of the calling process. The arguments that need to be passed are as follows:

The first argument, shmid, is the identifier of the shared memory segment. This id is the shared memory identifier, which is the return value of shmget() system call.

The second argument, shmaddr, is to specify the attaching address. If shmaddr is NULL, the system by default chooses the suitable address to attach the segment. If shmaddr is not NULL and SHM_RND is specified in shmflg, the attach is equal to the address of the nearest multiple of SHMLBA (Lower Boundary Address). Otherwise, shmaddr must be a page aligned address at which the shared memory attachment occurs/starts.

The third argument, shmflg, specifies the required shared memory flag/s such as SHM_RND (rounding off address to SHMLBA) or SHM_EXEC (allows the contents of segment to be executed) or SHM_RDONLY (attaches the segment for read-only purpose, by default it is read-write) or SHM_REMAP (replaces the existing mapping in the range specified by shmaddr and continuing till the end of segment).This call would return the address of attached shared memory segment on success and -1 in case of failure. To know the cause of failure, check with errno variable or perror() function.

Shmdt():

#include <sys/types.h>

#include <sys/shm.h>

int shmdt(const void *shmaddr)

The above system call performs shared memory operation for System V shared memory segment of detaching the shared memory segment from the address space of the calling process. The argument that needs to be passed is –

The argument, shmaddr, is the address of shared memory segment to be detached.

The to-be-detached segment must be the address returned by the shmat() system call .This call would return 0 on success and -1 in case of failure. To know the cause of failure, check with errno variable or perror() function.

Shmctl():

#include <sys/ipc.h>

#include <sys/shm.h>

int shmctl(int shmid, int cmd, struct shmid_ds *buf);

The above system call performs control operation for a System V shared memory segment. The following arguments needs to be passed −

The first argument, shmid, is the identifier of the shared memory segment. This id is the shared memory identifier, which is the return value of shmget() system call.

The second argument, cmd, is the command to perform the required control operation on the shared memory segment.

Valid values for cmd are −

IPC_STAT − Copies the information of the current values of each member of struct shmid_ds to the passed structure pointed by buf. This command requires read permission to the shared memory segment.

IPC_SET − Sets the user ID, group ID of the owner, permissions, etc. pointed to by structure buf.

IPC_RMID − Marks the segment to be destroyed. The segment is destroyed only after the last process has detached it.

IPC_INFO − Returns the information about the shared memory limits and parameters in the structure pointed by buf.

SHM_INFO − Returns a shm_info structure containing information about the consumed system resources by the shared memory.

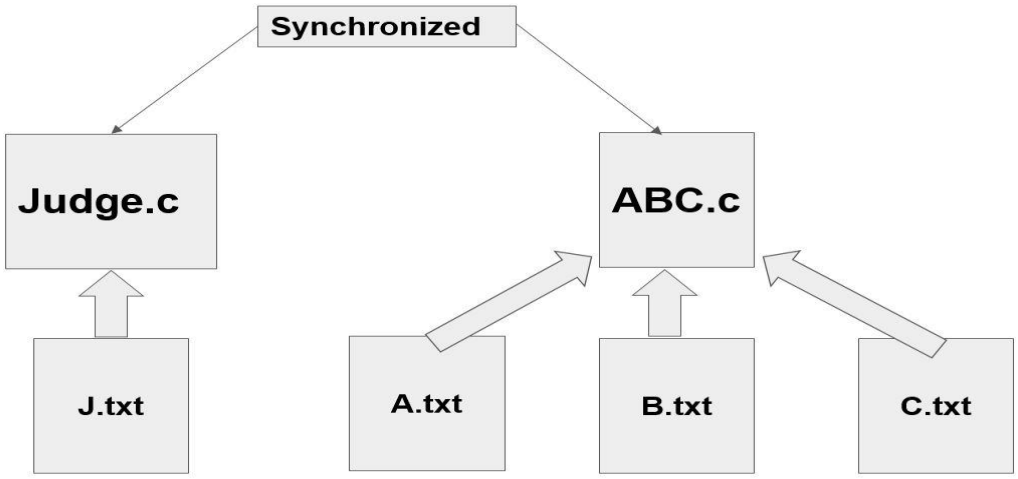
The third argument, buf, is a pointer to the shared memory structure named struct shmid_ds. The values of this structure would be used for either set or get as per cmd.

This call returns the value depending upon the passed command. Upon success of IPC_INFO and SHM_INFO or SHM_STAT returns the index or identifier of the shared memory segment or 0 for other operations and -1 in case of failure. To know the cause of failure, check with errno variable or perror() function.

//

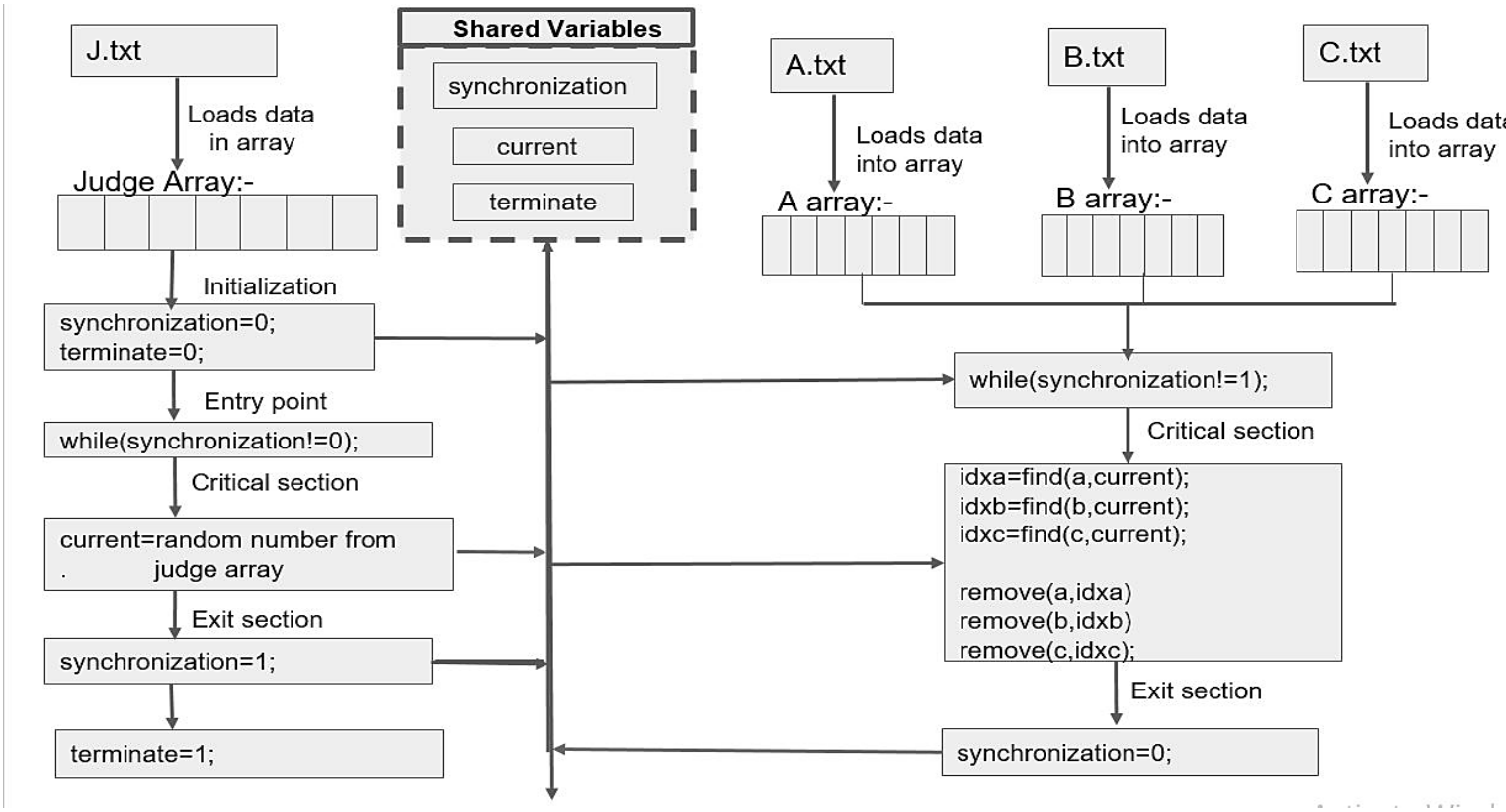
Implementation Description:

-Basic Structure



//

-code logic



//

-Explanation

Shared Variables:-
Current,synchronization and terminate.

Working:-
There are two C code files which share the above mentioned variables. The other four files are txt files which passes the array elements.
Judge file throughs random elements and those numbers are matched with arrA,arrB and arrC elements . These arrays represent different contestants playing housy .The elements are basically their tokens which gets exhausted . If the thrown element matches the array elements then that element its marked as visited and is not counted again.
The array whose all elements are visited first that member wins and accordingly the others win in sequence.
Working of shared variables:-
synchronization- These variables look at the synchronization part as one element is passed to contestants any other element must not go at the same time and bounded wait should occur.
terminate- It terminate the program when all contestant gets access to the pizza.
current- Current is the variable which is passed to arrays (arrA,aarB,aarC).

//

Program :

Judge.c

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/ipc.h>
#include<sys/shm.h>
#include<unistd.h>
#include<time.h>
int main()
{
   ///variables
   int size=0;//size of the array in file.
   int judge[10];//initialize  judge array
   int pass;
   int num=0;



   //opening judge file
   FILE *fpj;
   fpj=fopen("J.txt","r");

   //extracting data from file to array
   if(fpj==NULL)
   {
      printf("Couldn't open judge file\n");
   }
   else
   {
      int i=0;
      int buff;
      while(fscanf(fpj,"%d ",&buff)==1)
      {
         judge[i]=buff;
         size++;
         i++;
      }
      judge[i+1]=-1;//last element to -1 for end
      size=size+1;

   }
```

```
///shared part//////

//shared variables
key_t key1 = 5000;//key for current
key_t key2 = 6000;//key for synchronisation
key_t key3 = 7000;//key for terminate


int shmid1 = shmget(key1,4,0666|IPC_CREAT);//segment for current
int shmid2 = shmget(key2,4,0666|IPC_CREAT);//segment for synchronization
int shmid3 = shmget(key3,4,0666|IPC_CREAT);//segment for termination


int *current= (int*)shmat(shmid1,(void*)0,0);//for current element in judge
int *synchronization=(int*)shmat(shmid2,(void*)0,0);//for synchronization element in judge
int *terminate=(int*)shmat(shmid3,(void*)0,0);//for synchronization element in judge
*terminate=0;




//entry code

*synchronization=0;
pass=rand()%(size-1);
while(1){


   while(*synchronization!=0);




   ///critical section



   if(num<size-1){

      *current=judge[pass];
      judge[pass]=-999;

      printf("\nNumber is %d\n",*current);
      srand(time(NULL));

      num++;

      while(num<size-1){

         pass=rand()%(size-1);
```

```
            if(judge[pass]!=-999){
               break;
            }

         }



       }
      else{
         *terminate=1;
         *synchronization=1;
         exit(0);
      }



      ///exit section


      *synchronization=1;
      sleep(6);
}


   return 0;
}
```

//

ABC.c

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/ipc.h>
#include<sys/shm.h>
#include<unistd.h>
void rmv(int a[],int index);
void displaya(int a[],int n);
void displayb(int a[],int n);
void displayc(int a[],int n);
int iscomplete(int a[],int n);
int search(int a[],int n,int ele);
int main()
{

   FILE *fpa;


   // Opening first file of A
   fpa=fopen("A.txt","r");
   int sizea=0;              //Size of array of A
   int arrA[10];
```

```
    //extracting data from file to array
    if(fpa==NULL)
    {
        printf("Couldn't open judge file\n");
    }
    else
    {
        int i=0;
        int buff;
        while(fscanf(fpa,"%d ",&buff)==1)
        {
            arrA[i]=buff;
            sizea++;
            i++;
        }
        arrA[i+1]=-1;   //Last element to -1 for end
        sizea=sizea+1;

    }
//B
 FILE *fpb;
     // Opening first file of A
    fpb=fopen("B.txt","r");
    int sizeb=0;
    int arrB[10];

    //extracting data from file to array
    if(fpb==NULL)
    {
        printf("Couldn't open judge file\n");
    }
    else
    {
        int i=0;
        int buff;
        while(fscanf(fpb,"%d ",&buff)==1)
        {
            arrB[i]=buff;
            sizeb++;
            i++;
        }
        arrB[i+1]=-1; //last element to -1 for end
        sizeb=sizeb+1;
    }
//C
 FILE *fpc;
    fpc=fopen("C.txt","r");
    int sizec=0;
    int arrC[10];
```

```
//extracting data from file to array
if(fpc==NULL)
{
   printf("Couldn't open judge file\n");
}
else
{
   int i=0;
   int buff;
   while(fscanf(fpc,"%d ",&buff)==1)
   {
      arrC[i]=buff;
      sizec++;
      i++;
   }
   arrC[i+1]=-1;   //last element to -1 for end
   sizec=sizec+1;
}


///shared part//////

//shared variables
key_t key1 = 5000;//key for current
key_t key2 = 6000;//key for synchronisation
key_t key3 = 7000;//key for terminate

int shmid1 = shmget(key1,4,0666|IPC_CREAT);//segment for current
int shmid2 = shmget(key2,4,0666|IPC_CREAT);//segment for synchronization
int shmid3 = shmget(key3,4,0666|IPC_CREAT);//segment for termination
int *terminate=(int*)shmat(shmid3,(void*)0,0);//for synchronization element in judge
*terminate=0;

int *current= (int*)shmat(shmid1,(void*)0,0);//for current element in judge
int *synchronization=(int*)shmat(shmid2,(void*)0,0);//for synchronization element in judge


int printa=0,printb=0,printc=0;

//entry code
while(1){




while(*synchronization!=1);
//critical section

//check for termination
if(*terminate==1)
   exit(0);
```

```
int indexa,indexb,indexc;


int curr=*current;

printf("\nNew number came is %d\n",curr);



   indexa=search(arrA,sizea-1,curr);
   if(indexa!=-1)
      rmv(arrA,indexa);
   if(!iscomplete(arrA,sizea-1)){

      displaya(arrA,sizea-1);
   }
   else
      if(printa==0){
         printf("\nA got access to pizza\n");
         printa++;
      }



   indexb=search(arrB,sizeb-1,curr);
   if(indexb!=-1)
      rmv(arrB,indexb);
   if(!iscomplete(arrB,sizeb-1)){

      displayb(arrB,sizeb-1);
   }
   else
      if(printb==0){
         printf("\nB got access to pizza\n");
         printb++;
      }



   indexc=search(arrC,sizec-1,curr);
   if(indexc!=-1)
      rmv(arrC,indexc);
   if(!iscomplete(arrC,sizec-1)){

      displayc(arrC,sizec-1);
   }
```

```
        else
            if(printc==0){

                printf("\nC got access to pizza\n");
                printc++;
            }
//exit section

*synchronization=0;


sleep(3);


}



return 0;
}




//function to search specific element
int search(int a[],int n,int ele){

    for(int i=0;i<n;i++)
            if(a[i]==ele)
                    return i;

    return -1;
}

//fuction to update values
void rmv(int a[],int index){

    a[index]=-99;
}


void displaya(int a[],int n){

    printf("\n     ----------------------\n");
    printf("A-->|");
    for(int i=0;i<n;i++)
        if(a[i]==-99)
            printf("   |");
        else
            printf(" %d |",a[i]);
    printf("\n     ----------------------\n");
```

```
}
void displayb(int a[],int n){

   printf("\n     ----------------------\n");
   printf("B-->|");
   for(int i=0;i<n;i++)
      if(a[i]==-99)
         printf("   |");
      else
         printf(" %d |",a[i]);
   printf("\n     ----------------------\n");

}
void displayc(int a[],int n){

   printf("\n     ----------------------\n");
   printf("C-->|");
   for(int i=0;i<n;i++)
      if(a[i]==-99)
         printf("   |");
      else
         printf(" %d |",a[i]);
   printf("\n     ----------------------\n");

}

int iscomplete(int a[],int n){

   for(int i=0;i<n;i++)
         if(a[i]!=-99)
               return 0;

   return 1;
}
```

//

J.txt

```
13 2 1 3 4 5 6
```

A.txt

```
1 2 3 4 5
```

B.txt

```
2 3 4 5 13
```

C.txt

```
2 3 4 6 1
```

Input & Output :

Output 1:-

```
vishal@vishal-VirtualBox:~/Desktop/Assignment$ gcc Judge.c
vishal@vishal-VirtualBox:~/Desktop/Assignment$ ./a.out

Number is 2

Number is 3

Number is 4

Number is 13

Number is 6

Number is 1

Number is 5
vishal@vishal-VirtualBox:~/Desktop/Assignment$
```

```
vishal@vishal-VirtualBox:~/Desktop/Assignment$ gcc ABC.c
vishal@vishal-VirtualBox:~/Desktop/Assignment$ ./a.out

New number came is 2

     ----------------------
A-->| 1 |   | 3 | 4 | 5 |
     ----------------------

     ----------------------
B-->|   | 3 | 4 | 5 | 13 |
     ----------------------

     ----------------------
C-->|   | 3 | 4 | 6 | 1 |
     ----------------------

New number came is 3

     ----------------------
A-->| 1 |   |   | 4 | 5 |
     ----------------------

     ----------------------
B-->|   |   | 4 | 5 | 13 |
     ----------------------

     ----------------------
C-->|   |   | 4 | 6 | 1 |
     ----------------------
```

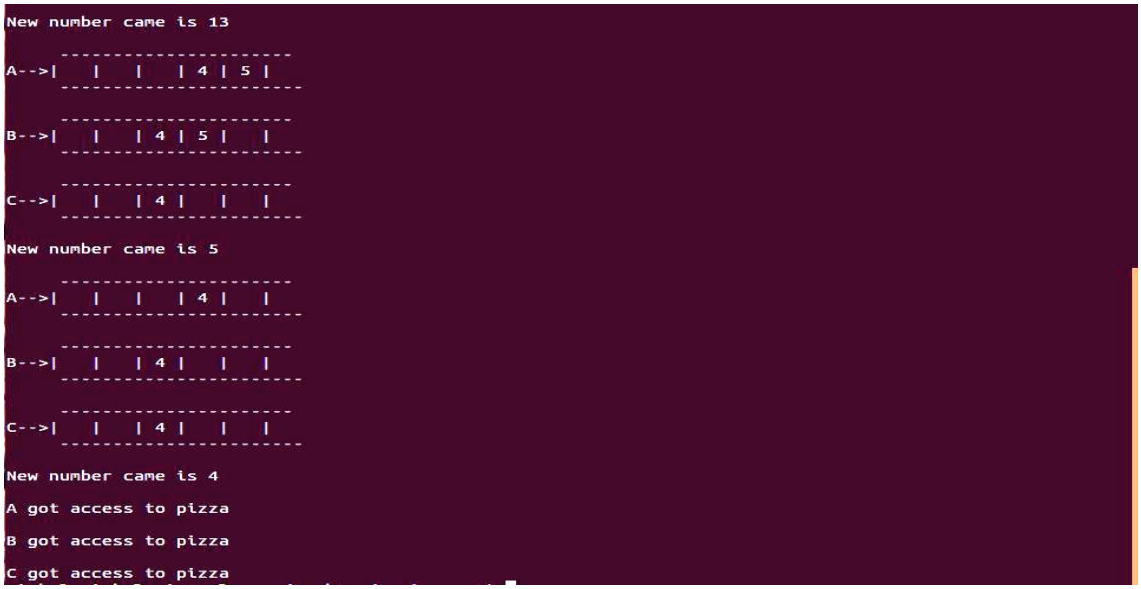
```
New number came is 6

     ----------------------
A-->| 1 |   |   |   | 5 |
     ----------------------

     ----------------------
B-->|   |   |   | 5 |   |
     ----------------------

     ----------------------
C-->|   |   |   |   | 1 |
     ----------------------

New number came is 1

     ----------------------
A-->|   |   |   |   | 5 |
     ----------------------

     ----------------------
B-->|   |   |   | 5 |   |
     ----------------------

C got access to pizza

New number came is 5

A got access to pizza

B got access to pizza
vishal@vishal-VirtualBox:~/Desktop/Assignment$
```

Output 2:-

```
vishal@vishal-VirtualBox:~/Desktop/Assignment$ gcc Judge.c
vishal@vishal-VirtualBox:~/Desktop/Assignment$ ./a.out

Number is 2

Number is 6

Number is 1

Number is 3

Number is 13

Number is 5

Number is 4
```

```
vishal@vishal-VirtualBox:~/Desktop/Assignment$ gcc ABC.c
vishal@vishal-VirtualBox:~/Desktop/Assignment$ ./a.out

New number came is 2

     ----------------------
A-->| 1 |   | 3 | 4 | 5 |
     ----------------------

     ----------------------
B-->|   | 3 | 4 | 5 | 13 |
     ----------------------

     ----------------------
C-->|   | 3 | 4 | 6 | 1 |
     ----------------------

New number came is 6

     ----------------------
A-->| 1 |   | 3 | 4 | 5 |
     ----------------------

     ----------------------
B-->|   | 3 | 4 | 5 | 13 |
     ----------------------

     ----------------------
C-->|   | 3 | 4 |   | 1 |
     ----------------------
```

```
New number came is 1

     ----------------------
A-->|   |   | 3 | 4 | 5 |
     ----------------------

     ----------------------
B-->|   | 3 | 4 | 5 | 13 |
     ----------------------

     ----------------------
C-->|   | 3 | 4 |   |   |
     ----------------------

New number came is 3

     ----------------------
A-->|   |   |   | 4 | 5 |
     ----------------------

     ----------------------
B-->|   |   | 4 | 5 | 13 |
     ----------------------

     ----------------------
C-->|   |   | 4 |   |   |
     ----------------------
```


New number came is 13
A-->| | | | 4 | 5 |
B-->| | | 4 | 5 | |
C-->| | | 4 | | |
New number came is 5
A-->| | | | 4 | | |
B-->| | | 4 | | |
C-->| | | 4 | | |
New number came is 4
A got access to pizza
B got access to pizza
C got access to pizza