The Implementation of Banker's Algorithm

Harshitha Yerraguntla(013800280)

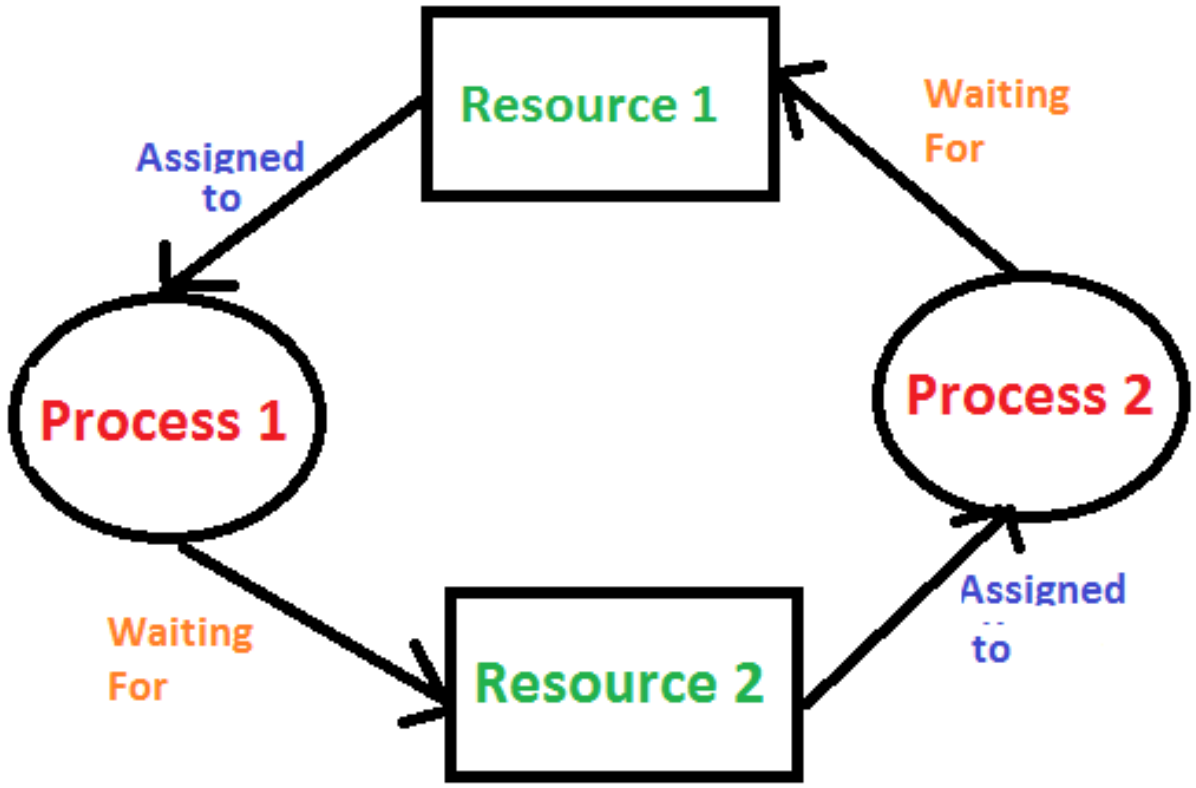
Nithya Kuchadi(013769665)

Koushik Kumar Kamala(013766571)

INDEX

DeadLock

- Process 1 is holding Resource 1 and waiting for resource 2 which is acquired by process 2, and process 2 is waiting for resource 1.



Four Conditions for Deadlocks

- **Mutual Exclusion:** At least one resource should be in non-shareable mode
- **Hold and Wait:** A process must be simultaneously holding at least one resource and waiting for at least one resource that is currently being held by some other process.
- **No Preemption:** Once a process is holding a resource, then that resource cannot be taken away from that process until the process voluntarily releases it.
- **Circular Wait:** A set of processes { P0, P1, P2, . . ., PN } must exist such that every P[i] is waiting for P[(i + 1) % (N + 1)]

Safe State vs Unsafe State

Safe state

- A state is safe if the system can allocate all resources requested by all processes (Upto Max) without entering a deadlock state.
- Safe Sequence: The order in which all processes can complete tasks using available resources
- If you find the safe sequence, then we can call it is Safe state

Unsafe State

- The state that a deadlock can occur
- Cannot find the Safe Sequence
- .All safe states are deadlock free, but not all unsafe states lead to deadlocks

Strategies for handling deadlocks

- **Deadlock prevention:** Prevents deadlocks by restraining requests made to ensure that at least one of the four deadlock conditions cannot occur.
- **Deadlock avoidance:** Dynamically grants a resource to a process if the resulting state is safe. A state is safe if there is at least one execution sequence that allows all processes to run to completion.
- **Deadlock detection and recovery:** Allows deadlocks to form; then finds and breaks them.

Bankers Algorithm

- Banker’s Algorithm is an example for Deadlock Avoidance
- It is used to determine whether a process’s request for allocation of resources be safely granted immediately.
- Each process has to specify its maximum requirement of resources in advance.
- A process is admitted for execution only if its need of resources is within the Available and Maximum capacity of resources.

Banker’s algorithm - Example

Process	Allocated				Max				Available			
	R1	R2	R3	R4	R1	R2	R3	R4	R1	R2	R3	R4
P1	0	0	1	2	0	0	1	2	2	1	0	0
P2	2	0	0	0	2	7	5	0				
P3	0	0	3	4	6	6	5	0				
P4	2	3	5	4	4	3	5	6				
P5	0	3	3	2	0	6	5	2				

Compute NEED Matrix

Process	Allocated				Max				Need				Available			
	R1	**R2**	**R3**	**R4**	**R1**	**R2**	**R3**	**R4**	**R1**	**R2**	**R3**	**R4**	**R1**	**R2**	**R3**	**R4**
P1	0	0	1	2	0	0	1	2	0	0	0	0	2	1	0	0
P2	2	0	0	0	2	7	5	0	0	7	5	0				
P3	0	0	3	4	6	6	5	0	6	6	2	2				
P4	2	3	5	4	4	3	5	6	2	0	0	0				
P5	0	3	3	2	0	6	5	2	0	3	2	0				

Is the system in Safe State?

Available = {2,1,0,0}

Iteration 1

P1 requests for resource

if (**P1 Need < Avail**)→TRUE

Availalble = Available + Allocated = } = **{2,1,1,2}**

For P2:→

if (**P2 Need < Avail**)→FALSE //then Check for next process.

For P3:→

if (**P3 Need < Avail**)→ FALSE //then Check for next process.

Is the system in Safe State?

- For P4:→

  if (**P4 Need < Avail**)→TRUE

  Avail= Avail + Allocated [P4] = **{4,4,6,6}**

- For P5:→

  if (**P5 Need < Avail**)→TRUE

  Avail= Avail + Allocated [P5] = = **{4,7,9,8}**

Is the system in Safe State?

Iteration 2. Check only process P2 to P3.

For P2:→

if (**P2 Need < Avail**)→TRUE

Available= Available + Allocated [P2] = **{6,7,9,8}**

For P3:→

if (**P3 Need < Avail**)→TRUE

Available= Available + Allocated [P3] = **{6,7,12,12} =System Capacity**

Safe Sequence

Since, all the processes got TRUE marked, no further iterations are required.

Safe Sequence = P1, P4, P5, P2 , P3

Data Structures used in Bankers Algorithm

- NUMBER_OF_CUSTOMERS = 5
- NUMBER_OF_RESOURCES = 3
- **available[NUMBER_OF_RESOURCES]:** 1-d array indicating the number of available resources of each type.

  Available[j] = k means there are **'k'** instances of resource type $\mathbf{R_j}$
- **maximum[NUMBER_OF_CUSTOMERS][NUMBER_OF_RESOURCES] :** 2-d array indicating the maximum demand of each process in a system

  Max[i, j] = k means Ci customer may request at most **'k'** instances of resource type $\mathbf{R_j}$.
- **allocation[NUMBER_OF_CUSTOMERS][NUMBER_OF_RESOURCES]:** 2-d array that defines the number of resources of each type currently allocated to each customer.

  Allocation[i, j] = k means Customer $\mathbf{C_i}$ is currently allocated **'k'** instances of resource type $\mathbf{R_j}$
- **need[NUMBER_OF_CUSTOMERS][NUMBER_OF_RESOURCES]:** 2-d array that indicates the remaining resource need of each customer.

  Need [i, j] = k means Customer $\mathbf{C_i}$ currently need **'k'** instances of resource type $\mathbf{R_j}$ for its execution.

Implementation

```
int main(int argc, const char * argv[])

{
for(int i=0 ;i<argc-1;i++){
        available[i]=atoi(argv[i+1]);
        total[i]=available[i];
    }
    for(int i=0;i< NUMBER_OF_RESOURCES;i++){
        for(int j=0;j<NUMBER_OF_CUSTOMERS;j++){
            maximum[j][i]=rand()%(total[i]+1);
            need[j][i]=maximum[j][i];}}
for (int i=0; i<NUMBER_OF_RESOURCES; i++) {
        string[i*2]=i+'A';
        string[i*2+1]=' ';
    }
    printf("Total system resources are:\n");
    printf("%s\n",string);
    for (int i=0; i<NUMBER_OF_RESOURCES; i++) {
        printf("%d ",total[i]);
    }
    printf("\n\nProcesses (maximum resources):\n");
    printf("   %s\n",string);

 for(int i=0; i<NUMBER_OF_CUSTOMERS;i++){
        printf("P%d ",i+1);
```

```
 for(int j=0;j<NUMBER_OF_RESOURCES;j++){
          printf("%d ",maximum[i][j]);
       }
       printf("\n");
   }
   printState();
   pthread_mutex_init(&mutex, NULL);
   pthread_t p1,p2,p3,p4,p5;
   int a=0,b=1,c=2,d=3,e=4;
   pthread_create(&p1,NULL,thread_func,&a);
   pthread_create(&p2,NULL,thread_func,&b);
   pthread_create(&p3,NULL,thread_func,&c);
   pthread_create(&p4,NULL,thread_func,&d);
   pthread_create(&p5,NULL,thread_func,&e);
   char *returnV;
   pthread_join(p1,(void**)&returnV);
   pthread_join(p2,(void**)&returnV);
   pthread_join(p3,(void**)&returnV);
   pthread_join(p4,(void**)&returnV);
   pthread_join(p5,(void**)&returnV);
   return 0;
}
```

```
void *thread_func(void* Tcustomer_num){
    int *c=(int*)Tcustomer_num;
    int customer_num= *c;
    int requestSum=0;
    while(!Finish[customer_num]){
        requestSum=0;
        int request[NUMBER_OF_RESOURCES]={0};
        for(int i=0;i<NUMBER_OF_RESOURCES;i++){
           request[i]=rand()%(need[customer_num][i]+1);
            requestSum=requestSum+request[i];
        }
        if(requestSum!=0)
            while(request_resources(customer_num,request)==-1);
}
return 0;
}
```

```
int request_resources(int customer_num, int request[]){
int returnValue=-1;
pthread_mutex_lock(&mutex);
printf("\nP%d requests for ",customer_num+1);
for(int i=0;i<NUMBER_OF_RESOURCES;i++){
       printf("%d ",request[i]);
   }
   printf("\n");
for(int i=0;i<NUMBER_OF_RESOURCES;i++){
    if(request[i]>available[i]){
          printf("P%d is waiting for the resources...\n",customer_num+1);
          pthread_mutex_unlock(&mutex);
          return -1;
       }
 }
returnValue=bankerAlgorithm(customer_num,request);
if(returnValue==0){
       int needIsZero=1;
       printf("a safe sequence is found: ");
       for(int i=0;i<NUMBER_OF_CUSTOMERS;i++){
          printf("P%d ",safeSequence[i]+1 );

       }
       printf("\nP%d's request has been granted\n",customer_num+1);
```

```
for(int j=0;j<NUMBER_OF_RESOURCES;j++){
        allocation[customer_num][j]=allocation[customer_num][j]+request[j];
        available[j]=available[j]-request[j];
        need[customer_num][j]=need[customer_num][j]-request[j];
        if(need[customer_num][j]!=0){
           needIsZero=0;
        }
 }
if(needIsZero){
        Finish[customer_num]=1;
        release_resources(customer_num);
}
printState();
}
   else{
      printf("cannot find a safe sequence\n");

   }
pthread_mutex_unlock(&mutex);
   return returnValue;
}
```

```
int release_resources(int customer_num){
    printf("P%d releases all the resources\n",customer_num+1);
    for(int j=0;j<NUMBER_OF_RESOURCES;j++){
        available[j]=available[j]+allocation[customer_num][j];
        allocation[customer_num][j]=0;
}
 return 0;
}
int bankerAlgorithm(int customer_num,int request[]){
    int finish[NUMBER_OF_CUSTOMERS]={0};
   for(int i=0;i<NUMBER_OF_RESOURCES;i++){
        Bavailable[i]=available[i];
        for(int j=0;j<NUMBER_OF_CUSTOMERS;j++){
            Ballocation[j][i]=allocation[j][i];
            Bmaximum[j][i]=maximum[j][i];
            Bneed[j][i]=need[j][i];
}}
for(int i=0;i<NUMBER_OF_RESOURCES;i++){
    Bavailable[i]=Bavailable[i]-request[i];
    Ballocation[customer_num][i]=Ballocation[customer_num][i]+request[i];
    Bneed[customer_num][i]=Bneed[customer_num][i]-request[i];
}
```

```
 int count=0;
 while(1){
 int I=-1;
for(int i=0;i<NUMBER_OF_CUSTOMERS;i++){
          int nLessThanA=1;
          for(int j=0; j<NUMBER_OF_RESOURCES;j++){
             if(Bneed[i][j]>Bavailable[j] || finish[i]==1){
                nLessThanA=0;
                break; } }
          if(nLessThanA){
             I=i;break; }}
 if(I!=-1){
          safeSequence[count]=I;
          count++;
          finish[I]=1;
          for(int k=0;k<NUMBER_OF_RESOURCES;k++){
             Bavailable[k]=Bavailable[k]+Ballocation[I][k];
          } }
       else{ for(int i=0;i<NUMBER_OF_CUSTOMERS;i++){
             if(finish[i]==0){
                return -1} }
          return 0; }
 }
}
```

```
void printState(){

   printf("Processes (currently allocated resources):\n");
   printf("   %s\n",string);
    for(int i=0; i<NUMBER_OF_CUSTOMERS;i++){
       printf("P%d ",i+1);
       for(int j=0;j<NUMBER_OF_RESOURCES;j++){
       printf("%d ",allocation[i][j]);} printf("\n") }
       printf("Processes (possibly needed resources):\n");
       printf("   %s\n",string);
   for(int i=0; i<NUMBER_OF_CUSTOMERS;i++){
       printf("P%d ",i+1);
       for(int j=0;j<NUMBER_OF_RESOURCES;j++){
           printf("%d ",need[i][j]); }printf("\n");
   }
   printf("Available system resources are:\n");
   printf("%s\n",string);
   for (int i=0; i<NUMBER_OF_RESOURCES; i++) {
       printf("%d ",available[i]);
   }

 printf("\n");

}
```

Output

```
Total system resources are:
A B C
10 5 7

Processes (maximum resources):
   A B C
P1 10 2 0
P2 1 0 5
P3 0 2 4
P4 6 5 2
P5 8 1 3
Processes (currently allocated resources):
   A B C
P1 0 0 0
P2 0 0 0
P3 0 0 0
P4 0 0 0
P5 0 0 0
Processes (possibly needed resources):
   A B C
P1 10 2 0
P2 1 0 5
P3 0 2 4
P4 6 5 2
P5 8 1 3
Available system resources are:
A B C
10 5 7

P1 requests for 7 1 0
a safe sequence is found: P1 P2 P3 P4 P5
P1's request has been granted
Processes (currently allocated resources):
   A B C
P1 7 1 0
P2 0 0 0
P3 0 0 0
P4 0 0 0
P5 0 0 0
Processes (possibly needed resources):
   A B C
P1 3 1 0
P2 1 0 5
P3 0 2 4
P4 6 5 2
P5 8 1 3
Available system resources are:
A B C
3 4 7

P1 requests for 1 0 0
a safe sequence is found: P1 P2 P3 P4 P5
P1's request has been granted
Processes (currently allocated resources):
   A B C
P1 8 1 0
P2 0 0 0
P3 0 0 0
P4 0 0 0
P5 0 0 0
Processes (possibly needed resources):
   A B C
P1 2 1 0
P2 1 0 5
P3 0 2 4
P4 6 5 2
P5 8 1 3
Available system resources are:
A B C
2 4 7

P1 requests for 1 0 0
a safe sequence is found: P1 P2 P3 P4 P5
P1's request has been granted
Processes (currently allocated resources):
   A B C
P1 9 1 0
P2 0 0 0
P3 0 0 0
P4 0 0 0
P5 0 0 0
```

```
Processes (possibly needed resources):
   A B C
P1 1 1 0
P2 1 0 5
P3 0 2 4
P4 6 5 2
P5 8 1 3
Available system resources are:
A B C
1 4 7

P1 requests for 1 1 0
a safe sequence is found: P1 P2 P3 P4 P5
P1's request has been granted
P1 releases all the resources
Processes (currently allocated resources):
   A B C
P1 0 0 0
P2 0 0 0
P3 0 0 0
P4 0 0 0
P5 0 0 0
Processes (possibly needed resources):
   A B C
P1 0 0 0
P2 1 0 5
P3 0 2 4
P4 6 5 2
P5 8 1 3
Available system resources are:
A B C
10 5 7

P5 requests for 0 0 3
a safe sequence is found: P1 P3 P4 P5 P2
P5's request has been granted
Processes (currently allocated resources):
   A B C
P1 0 0 0
P2 0 0 0
P3 0 0 0
P4 0 0 0
P5 0 0 3
```

```
Processes (possibly needed resources):
   A B C
P1 0 0 0
P2 1 0 5
P3 0 2 4
P4 6 5 2
P5 8 1 0
Available system resources are:
A B C
10 5 4

P5 requests for 3 1 0
a safe sequence is found: P1 P3 P5 P2 P4
P5's request has been granted
Processes (currently allocated resources):
   A B C
P1 0 0 0
P2 0 0 0
P3 0 0 0
P4 0 0 0
P5 3 1 3
Processes (possibly needed resources):
   A B C
P1 0 0 0
P2 1 0 5
P3 0 2 4
P4 6 5 2
P5 5 0 0
Available system resources are:
A B C
7 4 4

P5 requests for 1 0 0
a safe sequence is found: P1 P3 P5 P2 P4
P5's request has been granted
Processes (currently allocated resources):
   A B C
P1 0 0 0
P2 0 0 0
P3 0 0 0
P4 0 0 0
P5 4 1 3
Processes (possibly needed resources):
```