The Bankers Algorithm

First let's consider the situation when there is one resource type, think of it as units of money (1K dollars), a banker (the OS) who has a certain number of units in his bank and a number of customers who can loan a certain number of units from the bank and later pay the loan back (release the resources). The customers have a credit line that cannot exceed the number of units initially in the bank, and when they have borrowed their max number of units, they will pay their loan back.

The banker will grant a loan request only if it does not lead to an unsafe state. Eg, the banker initially has 10 K, and four customers A, B, C and D have credit lines 6K, 5K, 4K and 7K respectively. The state when no loans have been made is then:

```
   HAS MAX  Free:10
A  0   6
B  0   5
C  0   4
D  0   7
```

Say at some stage the loan situation is

```
   HAS MAX  Free:2
A  1   6
B  1   5
C  2   4
D  4   7
```

Suppose B loans one more unit:

```
   HAS MAX  Free:1
A  1   6
B  2   5
C  2   4
D  4   7
```

This is unsafe: if ALL CUSTOMERS ask their MAXIMUM remaining credit, NONE can be satisfied, and we have deadlock. So, in this case the banker will not grant B the loan; ie back to

```
   HAS MAX  Free:2
A  1   6
B  1   5
C  2   4
D  4   7
```

This state is safe because with 2K left, C can borrow her max remaining credit:

```
   HAS MAX  Free:0
A  1   6
B  1   5
C  4   4
D  4   7
```

finish and release her 4 units and finish

```
   HAS MAX  Free:4
A  1   6
B  1   5
D  4   7
```

The remaining state is safe. Why?

D can loan its 3 units

```
   HAS MAX  Free:1
A  1   6
B  1   5
D  7   7
```

and terminate

```
   HAS MAX  Free:8
A  1   6
B  1   5
```

Now A can loan its 5 units and terminate

```
   HAS MAX  Free:9

B  1   5
```

and now B can go

```
   HAS MAX  Free:10
```

So a safe state is a state where a sequence of ALL proceses can get their max required resources (one at the time) and finish and release all their resources.

MULTIPLE RESOURCE TYPES

The bankers algorithm for multiple resource types extends the method described above. Say we have five processes, 6 tape drives, 3 plotters, 4 printers, and 2 CD players. At some stage the state is

	Tp	Pl	Pr	CD	Tp	Pl	Pr	CD		Tp	Pl	Pr	CD	
A	3	0	1	1	1	1	0	0	E =	6	3	4	2	E: Existing resources
B	0	1	0	0	0	1	1	2	P =	5	3	2	2	P: Possessed resources
C	1	1	1	0	3	1	0	0	A =	1	0	2	0	A: Available resources
D	1	1	0	1	0	0	1	0						
E	0	0	0	0	2	1	1	0						

HAS STILL NEEDS

The Column sums of the HAS matrix is equal to the vector P. Also, E = P+A, or A = E-P.

Checking whether the state is safe.

1. Find a process (row) R with unmet resources ALL less than A, ie a process that can be granted all its unmet resources. If no such row exists, the state is unsafe.
2. Assume process R takes all its resources and finishes. Mark it as finished and release its resources back to A.
3. Repeat steps 1 and 2 until either an unsafe state appears, in which case there is a potential for deadlock, or all processes finish, in which case the original state was safe.

The state in the example above is safe, as D can get a Printer and finish:

	Tp	Pl	Pr	CD	Tp	Pl	Pr	CD		Tp	Pl	Pr	CD	
A	3	0	1	1	1	1	0	0	E =	6	3	4	2	E: Existing resources
B	0	1	0	0	0	1	1	2	P =	4	2	2	1	P: Possessed resources
C	1	1	1	0	3	1	0	0	A =	2	1	2	1	A: Available resources
E	0	0	0	0	2	1	1	0						

HAS STILL NEEDS

Then E can get all its resources and finish

```
Tp Pl Pr CD   Tp Pl Pr CD        Tp Pl Pr CD
 A  3  0  1  1    1  1  0  0      E =  6  3  4  2  E: Existing resources
 B  0  1  0  0    0  1  1  2      P =  4  2  2  1  P: Possessed resources
 C  1  1  1  0    3  1  0  0      A =  2  1  2  1  A: Available resources

      HAS       STILL NEEDS
```

Then B can go

```
  Tp Pl Pr CD   Tp Pl Pr CD        Tp Pl Pr CD
 A  3  0  1  1       1  1  0  0     E =  6  3  4  2  E: Existing resources
                                    P =  4  1  2  1  P: Possessed resources
 C  1  1  1  0       3  1  0  0     A =  2  1  2  1  A: Available resources

      HAS       STILL NEEDS
```

Then A can go

```
  Tp Pl Pr CD   Tp Pl Pr CD        Tp Pl Pr CD
                                    E =  6  3  4  2  E: Existing resources
                                    P =  1  1  1  0  P: Possessed resources
 C  1  1  1  0       3  1  0  0     A =  5  2  3  2  A: Available resources

      HAS       STILL NEEDS
```

And finally C can go

```
  Tp Pl Pr CD   Tp Pl Pr CD        Tp Pl Pr CD
                                    E =  6  3  4  2  E: Existing resources
                                    P =  0  0  0  0  P: Possessed resources
                                    A =  6  3  4  2  A: Available resources

      HAS       STILL NEEDS
```

Now what about deciding whether a request can be granted. Let's go back to the original state:

```
  Tp Pl Pr CD   Tp Pl Pr CD        Tp Pl Pr CD
 A  3  0  1  1       1  1  0  0     E =  6  3  4  2  E: Existing resources
 B  0  1  0  0       0  1  1  2     P =  5  3  2  2  P: Possessed resources
 C  1  1  1  0       3  1  0  0     A =  1  0  2  0  A: Available resources
 D  1  1  0  1       0  0  1  0
 E  0  0  0  0       2  1  1  0

      HAS       STILL NEEDS
```

Suppose B requests a printer. This would make the state:

	Tp	Pl	Pr	CD	Tp	Pl	Pr	CD					
A	3	0	1	1	1	1	0	0	E =	6	3	4	2 E: Existing resources
B	0	1	1	0	0	1	0	2	P =	5	3	3	2 P: Possessed resources
C	1	1	1	0	3	1	0	0	A =	1	0	1	0 A: Available resources
D	1	1	0	1	0	0	1	0					
E	0	0	0	0	2	1	1	0					

HAS STILL NEEDS

Which is safe, as
D can still finish

	Tp	Pl	Pr	CD	Tp	Pl	Pr	CD					
A	3	0	1	1	1	1	0	0	E =	6	3	4	2 E: Existing resources
B	0	1	1	0	0	1	0	2	P =	4	2	3	1 P: Possessed resources
C	1	1	1	0	3	1	0	0	A =	2	1	1	1 A: Available resources
E	0	0	0	0	2	1	1	0					

HAS STILL NEEDS

now E can go

	Tp	Pl	Pr	CD	Tp	Pl	Pr	CD					
A	3	0	1	1	1	1	0	0	E =	6	3	4	2 E: Existing resources
B	0	1	1	0	0	1	0	2	P =	4	2	3	1 P: Possessed resources
C	1	1	1	0	3	1	0	0	A =	2	1	1	1 A: Available resources

HAS STILL NEEDS

now A can go

	Tp	Pl	Pr	CD	Tp	Pl	Pr	CD					
					1	1	0	0	E =	6	3	4	2 E: Existing resources
B	0	1	1	0	0	1	0	2	P =	1	2	2	0 P: Possessed resources
C	1	1	1	0	3	1	0	0	A =	5	1	2	2 A: Available resources

HAS STILL NEEDS

now B can go

```
  Tp Pl Pr CD   Tp Pl Pr CD         Tp Pl Pr CD
                                E =  6  3  4  2  E: Existing resources
                                P =  1  1  1  0  P: Possessed resources
C  1  1  1  0         3  1  0  0   A =  5  2  3  2  A: Available resources

     HAS        STILL NEEDS
```

and finally C can go

```
  Tp Pl Pr CD   Tp Pl Pr CD         Tp Pl Pr CD
                                E =  6  3  4  2  E: Existing resources
                                P =  0  0  0  0  P: Possessed resources
                                A =  6  3  4  2  A: Available resources

     HAS        STILL NEEDS
```

So the bankers algorithm, upon receiving a request, "pencils" it into the state and checks whether the state is still safe.

Deadlock Detection

Deadlock occurs fairly rarely, so it is more efficient to check say once every 10 minutes for deadlock or when the CPU usage is low even though many processes are waiting, and then deal with it if, it has occurred.

Deadlock detection employs simplified versions of deadlock avoidance algorithms. In the case of a single resource per resource type we can create a simplified resource-allocation graph called the WAIT-FOR graph, and in the case of multiple resi=ource instances per type we can use the check part of the Banker's algorithm, without dealng with new requests.

Single Resource per Resource Type

If there is only a single resource per type and we only want to detect deadlock when it alkready heppened, we don't need th eresources tin the resource allocation graph. The graph just becomes a Pi waits for Pj (because Pj has the resource Pi needs). A cycle in that graph then indicates deadlock.

Multiple Resources per Resource Type

The multiple resource deadlock detection algorithm uses the HAS, STILL NEEDS matrices and the AVAILABLE resources vector. It just checks for the state to be safe by iteratively marking a process that has needs less than the available resouce vector and putting the resources that process has back into the available vector. Unmarked processes at the end of that check are in deadlock with each other.

	Allocation	Request	Available
	A B C	A B C	A B C
P0	0 1 0	0 0 0	0 0 0
P1	2 0 0	2 0 2	
P2	3 0 3	0 0 0	
P3	2 1 1	1 0 0	
P4	0 0 2		

P2 can finish because it has all the resources it needs.

	Allocation	Request	Available
	A B C	A B C	A B C
P0	0 1 0	0 0 0	3 0 3
P1	2 0 0	2 0 2	
P3	2 1 1	1 0 0	
P4	0 0 2		

Which process can get all the resources it needs to finish?

P0 also has all the resources it needs.

	Allocation	Request	Available
	A B C	A B C	A B C
P0			3 1 3
P1	2 0 0	2 0 2	
P3	2 1 1	1 0 0	
P4	0 0 2		

Which process is next?

	Allocation	Request	Available
	A B C	A B C	A B C
P0			3 1 3
P1	2 0 0	2 0 2	
P3	2 1 1	1 0 0	
P4	0 0 2		

Any of the remaining processes can get the resources it needs – so we are not in deadlock.

Let’s take the situation of :

	Allocation	Request	Available
	A B C	A B C	A B C
P0	0 1 0	0 0 0	0 0 0
P1	2 0 0	2 0 2	
P2	3 0 3	1 0 0	
P3	2 1 1	1 0 0	
P4	0 0 2		

P0 can finish

	Allocation	Request	Available
	A B C	A B C	A B C
P0			0 1 0
P1	2 0 0	2 0 2	
P2	3 0 3	1 0 0	
P3	2 1 1	1 0 0	
P4	0 0 2		

No one else can get their resources, so we are in deadlock.

Deadlock Recovery

We can either TERMINATE PROCESSES or PREEMPT RESOURCES and give them to other processes, as we did in deadlock prevention.

Process termination

Terminating all processes in a deadlock cycle is unnecessary, we can succssively terminate processes until the deadlock disappears (often we need to terminate only one process). We need to terminate the process that incurs the least cost, but the cost is not a precise measure. Here are some factors:

- process priority
- how long has the process already computed
- number of resources the process has
- number of resources a process can still request

Resource Preemption

Remember, preemption should only be applied to resources whose state can be easily saved and restored (registers, memory). Issues:

- Select resource and process to preempt. Again there are fuzzy cost measures.

- Rollback. Where to restart the process that had the preempted resource? We can put the process in the wait queue and only when the resource comes back restart in the context where the preemption occurred, or we can roll the process back to a state where it requested the resource,but that requires checkpointing, or we can completely restart the process. It is bad, but still better than rebooting the whole system.

- Starvation. Avoid rolling back the same process all the time, eg using a counter.