User-Space I/O Software

Most of I/O software is part of the OS.

A small portion consists of

- Libraries

  I/O system calls are typically made by library procedures, e.g. printf, which are linked into the program during compilation (static) or loading (dynamic).

- Programs that run outside of the OS (e.g. **system daemons**)

  Spooling is a way of dealing with dedicated I/O devices in a multiprogrammed system.

  - User generates text to be printed and places file in a special directory, spool directory, e.g. /usr/spool/lp
  - Printer daemon wakes up occasionally and checks directly. If a file exists, it starts sending it to the printer.

  This protects the device but allows users access to it.
  Similar mechanism exists for email.

3.3 Deadlocks

A system consists of a finite number of system **resources** to be distributed among a number of competing processes.

Resources can be partitioned into several types, e.g.,

- Memory
- CPU cycles
- Files
- I/O devices

There can be multiple instances of each type.

All OSs have ability to grant users exclusive rights to system resources, e.g. CDROM readers and recorders, 8mm DAT tape drives, etc.

Let us define a **resource** as anything (hardware or software) that can only be used by a single process at any instant.

A process must request a resource before using it, and must release the resource after using it.

A process may utilize a resource in only the following sequence:

- **Request**: If the request cannot be granted immediately, the process is blocked.
- **Use**: The process uses the resource (e.g., prints a file).
- **Release**: The process releases the resource.

The **Request** and **Release** of hardware and software resources are system calls, e.g., **open**/**close** file or device, **allocate**/**free** memory.

Other software resources include database records; request and release may be accomplished through **record locking**.

If a process requests a resource that is currently being used by another process, the OS adds the process to a wait queue.

OS keeps track of which processes are using which resources.

Many user processes require exclusive access to **more than one** resource.

The order in which a process requests resources and the OS grants access to these resources can result in **deadlock**, e.g.,

Process A requests and OS grants use of printer (context switch).
Process B requests and OS grants use of plotter (context switch).
Process A requests use of plotter. OS blocks process A.
Process B requests use of printer. OS blocks process B.
Both processes are blocked forever: **deadlock**.

Deadlock defined:

A set of processes is in a deadlock state when every process in the set is waiting for an event that can be caused by only another process in the set.

In most cases, the event that each process is waiting for is the release of some resource currently possessed by another member of the set.

Deadlocks can occur for hardware or software resources, e.g. database accesses can deadlock when processes lock records for updates, e.g.,

Record A is your checking account balance.
Record B is your savings account balance.

Process A	Process B
lock(record A)	lock(record B)
read A	read B
lock(record B)	lock(record A)
read B	read A
B = B+ 100	A = A + 100
A = A - 100	B = B - 100
update A	update B
update B	update A
unlock(B)	unlock(A)
unlock(A)	unlock(B)

With locks (or Waits), deadlock can occur.
Without them, walkie-talkies will sell like hotcakes.

Two types of resources

- **Preemptable resource**:
  A resource that can be taken away from a process owning it with no ill effects. e.g.,
  If the OS supports virtual memory, processes may be swapped in and out. Real memory is a preemptable resource.

  Deadlock can usually be resolved by reallocating resources.

- **Nonpreemptable resource**:
  A resource that can NOT be taken away from a process without causing the computation or operation to fail. e.g.,
  A process is printing a file. If printer is preempted from the process and given to another process, the result is garbage.

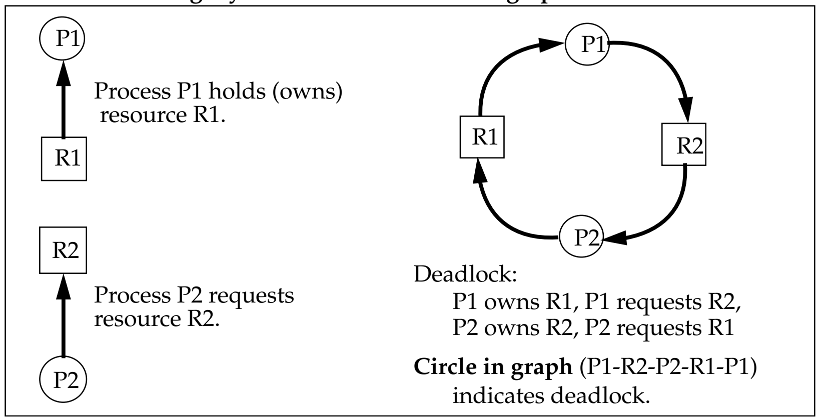
Resources protected with semaphores, e.g. the screen, are NOT included here. These resources are preemptable and must be protected with mutual exclusion mechanisms.
Nonpreemptable resources are problematic since they can result in deadlock.

Four conditions that MUST be present for deadlock to occur:

- **Mutual exclusion** condition: At least one resource must be held in a non-sharable mode; that is, only one process at a time can use the resource.
- **Hold and wait** condition: There must exist a process that is holding at least one resource and is waiting to acquire additional resources that are currently held by other processes.
- **No preemption** condition: Resources previously granted cannot be forcibly taken away from a process. Processes release resources voluntarily.
- **Circular wait** condition: There must be a circular chain of two or more processes, each of which is waiting for a resource held by the next member of the chain.

Note that the **Circular wait** condition implies the **Hold and wait** condition, so the four conditions are not completely independent.

Deadlock Modeling: System resource-allocation graph



Arcs from resource to process indicate process requested and was granted the resource.

Arcs from process to resource indicate process is requesting the resource.

Suppose we have three processes, A, B and C.
Suppose we have three resources, X, Y and Z.

Suppose each process requests resources as:

Process A: Request X, Request Y, Release X, Release Y.
Process B: Request Y, Request Z, Release Y, Release Z.
Process C: Request Z, Request X, Release Z, Release X.

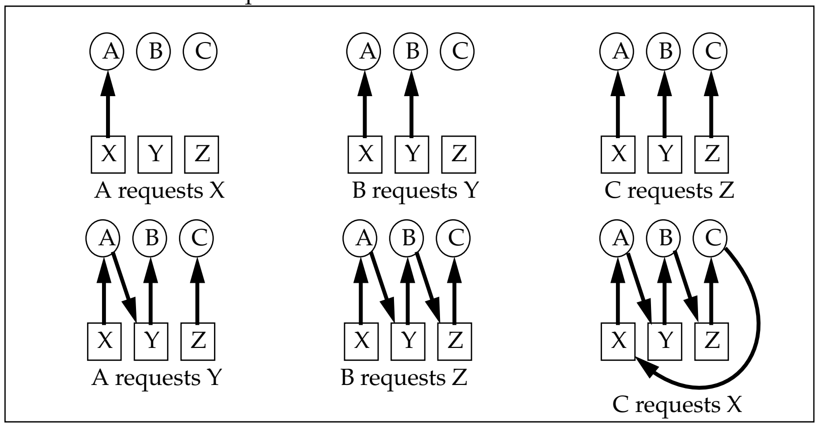
Suppose OS schedules processes to run sequentially as A, B and C and allows them to run to completion. Assume X, Y and Z are nonpreemptable resources.

Can this result in deadlock ? Which condition(s) does not hold ?

What is wrong with this scheduling algorithm ?

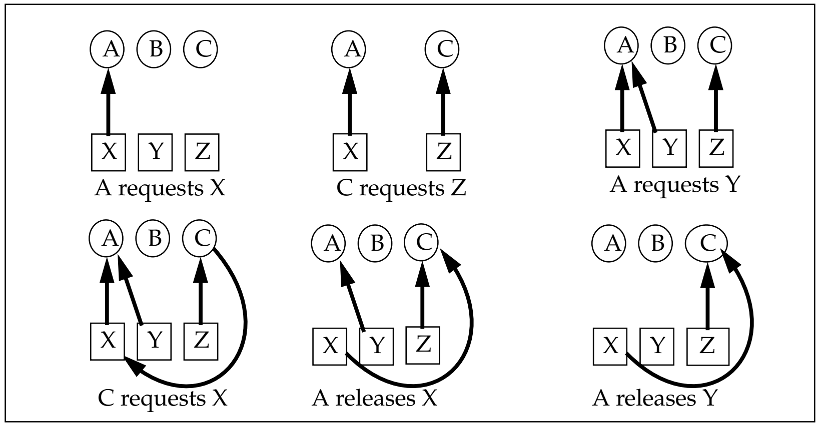
Let us assume that processes do both computing and I/O and the OS uses a round robin scheduling algorithm.

Assume resource requests occur as:



Are processes A, B and C deadlocked ? Why or why not ?

Suppose OS chose to run the processes in a different way.
Suppose OS knew about the impending deadlock and did not schedule process B until it knew it would be 'safe' to do so.
Using the same sequence of requests from A and C but suspending B results in:



Deadlock ?

4 strategies are used for dealing with deadlocks

- Ignore the problem altogether.
- Detection and recovery.
- Prevention, by structurally negating one of the four sections.
- Dynamic avoidance by careful resource allocation.

3.3.3 Ostrich Algorithm

Stick your head in the sand and pretend there is no problem at all.

Used in systems where the OS crashes for other reasons more often then it locks up because of deadlocks (Windows 95 ???).

Actually, UNIX uses this philosophy for the most part, e.g.,

Process slots (PCBs) are limited so 'forks' can fail.
Assume 10 running processes are trying to fork 12 subprocesses each, and the table slots are limited to 100.
If each process has forked 9 times, table is full.
Each of the 9 processes sits in a endless loop, forking and failing.
The system is deadlocked.

Ignoring the existence of a problem saves the overhead and/or the inconvenience of applying one of the other strategies described below.

3.3.4 Detection and Recovery

OS monitors requests and release of resources and updates resource graph.

If a cycle is detected, one of the processes in the cycle is killed.

This continues until the cycle is broken.

Simpler approach is to kill processes that have been continuously blocked for more than say, 1 hour. No graph is maintained.

However, this requires restoration of files to their original state and the undoing of other side effects.

- Difficult to undo file modification unless a copy is kept.
- Database transactions can be rolled back.

3.3.5 Deadlock Prevention

Idea: Remove one of the 4 conditions for deadlocks making their occurrence impossible.

Removing mutual exclusion and no preemption conditions are difficult, e.g.,

- Removing mutual exclusion to the printer or allowing preemption of the printer results in garbage.

- Spooling can be used to remove mutual exclusion condition.

- But spooling itself can lead to deadlock, e.g.,
  - Two processes could each fill up half of the space available to spool files.
  - Printer can NOT print either of the files until the entire file is present.
  - Deadlock !

- Preemption can be applied to resources whose state can be easily saved and restored later, such as CPU registers and memory space.

Condition 2, Hold and Wait:

- Idea:
  - Prevent processes that hold resources from waiting for more resources.
- Implementation:
  - Require all processes to request (and acquire) resources before starting execution (or to make system calls for resources before any other system calls).
- Problem:
  - Many processes do not know what resources they will require.

  - Moreover, resources are not used optimally, e.g., a process could request a read lock on a tape drive, compute something for an hour, and then write an output tape and plot the results.

  - This prevents other processes from using the output tape and plotter for an hour !

Condition 4, Circular Wait:

- Idea:
  - Enforce the rule that a process is entitled only to a single resource at any moment.
- Implementation:
  - If it needs a second resource, it MUST release the first one.
- Problem:
  - It will not be possible for a process, for example, to copy a huge file from a tape to a printer. Unacceptable.

Condition 4, Circular Wait (continued):

Idea:

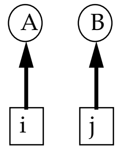
Globally number all resources and force processes to acquire resources in numerical order.

Implementation:

1) CD-ROM
2) Printer
3) Plotter
4) Tape drive
5) Robot arm

Processes can request the printer and then the plotter, but not in the reverse order.

This rule prevents cycles in the resource graph, and therefore, prevents deadlock, e.g.,



To get a deadlock, A must request j and B must request i.

Suppose i > j, then A can NOT obtain resource j.

Suppose i < j, then B can NOT obtain resource i.

Problem:

Since the number of potential resources is large, it may be impossible to find an ordering that satisfies everyone.

3.3.5 Deadlock Avoidance

By analyzing each resource request, the OS can determine if it can be safely granted.

Banker's algorithm: Dijkstra's scheduling algorithm that avoids deadlocks.

Requirements:

Each process must declare the maximum number of resources of each type that it may need.

The OS maintains information concerning:

- The number of existing resources of each type.
- The number of assigned resources of each type.
- The number of available resources of each type.

The algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition.

Modeled in the way a small-town banker might deal with a group of customers to whom he has granted lines of credit:

Initial state of the system

	Used	Maximum
Andy	0	6
Barbara	0	5
Marvin	0	4
Suzanne	0	7

Available: 10

Maximum: Customers credit line in Units of 1 K dollars. Sum is 22.

Available: (10) Less than sum of Maximums since not all customers will need their maximum immediately.

Assume

- Customers are processes
- Units are tape drives
- Banker is OS.

Assume some processes ran (customers made requests) and we examine the state of the system at some random time t_1.

New state of the system at some time $t_1 > t_0$.

	Used	Maximum
Andy	1	6
Barbara	1	5
Marvin	2	4
Suzanne	4	7

Available: 2

Safe State: Since there exists a sequence of other states that leads to all the customers getting loans up to their credit limits.

Why ? Because Marvin's request can be granted (others must be delayed) and allowed to finish.

This will cause all of his resources to be freed (now 4 available) and Suzanne or Barbara can finish, followed by Andy.

Assume some processes ran and we examine the state of the system at some random time t_2.

New state of the system at some time $t_2 > t_1$.

	Used	Maximum
Andy	1	6
Barbara	2	5
Marvin	2	4
Suzanne	4	7

Available: 1

UnSafe State: Barbara asked for one more Unit and was granted it.

Why ? Because if all customers suddenly asked for their maximum loans, the banker could not satisfy ANY of them - deadlock.

Note: An unsafe state does NOT have to lead to deadlock.

If Barbara released a Unit, the state would again be safe.

However, the banker can not count on this behavior.

The banker's algorithm considers each request as it occurs and determines if granting it leads to a safe state:

- If it does, the request is granted.
- If not, the request is postponed.

In this case, the banker checks to see if he has enough resources to satisfy the customer closest to his or her maximum.

If so, this loan is assumed to be repaid (resources are released).

The banker then checks the customer that is now closest to his or her limit.

If enough resources are available to satisfy the request, then this loan is assumed to be repaid, etc.

If all loans can eventually be repaid, the state is safe and the initial request can be granted.

Note: A customer may request a Unit that is available but is still made to wait. Therefore, resource utilization may be lower than it would be otherwise.

Generalized to multiple processes and multiple resources:

New state of the system at some time $t_1 > t_0$.

Process	Tape drives	Plotters	Printers	CD-ROMs
A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0

Resources assigned

Process	Tape drives	Plotters	Printers	CD-ROMs
A	1	1	0	0
B	0	1	1	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

Resources still needed

Existing = (6342)
Possessed = (5322)
Available = (1020)

Existing: 6 tape drives, 3 plotters, 4 printers and 2 CD-ROMs.

Possessed: 5 tape drives, 3 plotters, 2 printers and 2 CD-ROMs.
Sum of columns in left hand matrix.

Available = Existing - Possessed.

This state is safe. Why ? Because D can finish - which changes Available to (2121) - followed by A or E - A first (5132), E first (2121) than A (5132) followed by B or C - C first (6242), B first (5232).

Algorithm:

- Look for a row, R, whose unmet resource needs (right matrix) are all smaller than or equal to A.
  - If no such row exists, the system will eventually deadlock since no process can run to completion.

- Assume the process of the row found above requests all the resources it needs (which is guaranteed to be possible) and finishes.
  - Mark that process as terminated and add all its resources to the A vector.

- Repeat above steps until either:
  - All processes are marked terminated, in which case the initial state was safe, or
  - Until a deadlock occurs, in which case it was not.

Assume process B now requests a printer.

Process	Tape drives	Plotters	Printers	CD-ROMs
A	3	0	1	1
B	0	1	**1**	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0

Resources assigned

Process	Tape drives	Plotters	Printers	CD-ROMs
A	1	1	0	0
B	0	1	**0**	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

Resources still needed

Existing = (6342)
Possessed = (5332)
Available = (10**1**0)

This state is safe. Why ?
Because D can still finish followed by A or E, etc. - same as before.

Assume process E now requests a printer.

Process	Tape drives	Plotters	Printers	CD-ROMs
A	3	0	1	1
B	0	1	**1**	0
C	1	1	1	0
D	1	1	0	1
E	0	0	**1**	0

Resources assigned

Process	Tape drives	Plotters	Printers	CD-ROMs
A	1	1	0	0
B	0	1	**0**	2
C	3	1	0	0
D	0	0	1	0
E	2	1	**0**	0

Resources still needed

Existing = (6342)
Possessed = (5342)
Available = (10**0**0)

This state is NOT safe. Why ?
Because NO process can finish.

Assume process A requests 1 tape drive instead and needs 2 plotters.
D can still finish with Avail = (0010). But Avail becomes (1111) and A cannot finish (needs 2 plotters), E cannot finish (needs 2 tape drives), B cannot finish (needs 2 CD-ROMs), C cannot finish (needs 3 tape drives).

Problem:
- In theory, the algorithm is wonderful.
- In practice, it is essentially useless !!!

Why ?
- Processes rarely know what their maximum resource needs are in advance.
- The number of processes are not fixed but vary dynamically.
- Some resources can vanish - the printer could be stolen, a hard drive head could nose dive into a platter, the tape drive might self destruct (mission impossible), the data CD may be replaced with a ‘Guns and Roses’ CD.

Therefore,
- Deadlock prevention is overly restrictive.
- Deadlock avoidance is based on information that is usually not available.

Want to be famous ?
- Figure out a better way !