Kernel to User transfer can be triggered due to new processes, resume after an interrupt, processor exception, or system call, switch to a different process, user- level up call.

Interrupt:

An interrupt is a hardware signal from a device to a CPU. It tells the CPU that the device needs attention and the CPU should stop preforming what it is doing and respond to device. It can be masked.

Processor exception:

Faults, errors, such as try to divide by 0.

System call:

Request from user level to the kernel level, only can be made in the user level.

User-level upcall:

It is a mechanism that allow kernel to execute a function in the user level.

User to kernel transfer: Interrupts, exceptions, system calls

At a minimum, the hardware must support, privileged instructions, memory protection, timer interrupts.

Three roles an operating system plays include referee, illusionist, and glue.

A batch operating system works on a queue of task. It runs a simple loop: run and unload each job in turn.

Most operating systems allocate both a user stack and kernel stack for interrupted process.

A shell is a job control system

How windows create a process:

Create and initialized PCB, process control block.

Create and initialized a new address space

Load the program into the address space

Initialize the hardware context to start execution at "start"

Inform the scheduler that the new process is ready to run.

Thread operations:
Thread_create()

Thread_yield()

Thread_join()

Thread_exit()

Event Driven: An alternate approach to threading.

Thread life cycle:
init state, and then switch to ready state when put in the ready list, running state after being scheduled. At the running state, thread may terminate and switch to finished state, or suspend due to IO and switch to waiting state or yield the CPU and entered into ready state.

Steps to create thread:

Allocate per thread-state, initial per-thread state, and put PCB to ready list.

Two types of kernel thread context switch:

Voluntary:

Three thread operations

Involuntary:

Timer interrupt or exception

IO hardware events

Some other thread is higher priority.

Four states of thread:

Ready, Running, Waiting, Finished.

TCB: TCB includes, thread stack, copy of processor registers, per-thread metadata.

Unix fork does create new address space, create and initialize the process control block in the kernel, inherit the execution context of the parent.

Lock: A lock should ensure three properties, progress, mutual exclusion, bounded waiting.

Muti-thread process:

Muti-thread-process using kernel thread, User-level threads without kernel support, Muti-threaded process using kernel support.

Deadlock:

Four conditions:

No preemption

Bounded resources

Wait while holding

Circular waiting

Readers and wirters

```
StartRead()
{
  Lock.acquire()
  WaitingReaders++
  While(ReadShouldWait())
          ReadGo.wait(&lock)
  WatingReaders—
  ActiveReaders++
Lock.release()
}
```

```
DoneRead()
{
 Lock.acquire()
ActiveReaders—
If(activeReader==0  && waitingWriters >0)
            WriterGo.signal()
Lock.release()
}
Bool ReadShouldWait
{
    Return activeWrtiers>0 || waitingWriters>0
}
StartWrite()
{
      Lock.acquire()
      waitingWriters++
       while(writeShouldWait())
            writeGo.wait(&lock)
       waitingWriters—
        activeWriters++
       lock.release()
}
```

```
DoneWrite()
{
   Lock.acquire()
   activeWriter--
   assert (activeWriters ==0)
   if(waitingWriters>0)
        WriterGo.signal()
   else
        readGo.broadcast()
   Lock.release()
}
writeShouldWait
{
    activeWriters>0 || activeReaders>0
}
```

Bounded Buffer

```
Get ()
{
Lock.acqure()
While(front==tail)
        Empty.wait(lock)
Item = buff[front%max]
Front++
Full.signal(lock)
Lock.release()
Return Item
}
```

```
Put(item)
{
    Lock.acquire()
    While((tail-front) ==max)
        Full.wait(lock)
    Buf[tail%max] = item
    Tail++
    Empty.signal(lock)
    Lock.release()
}
```

Banker Algorithm

Resources: 8

Requirements: A 4 B 5 C 5

Current: A 3 B 2 B 2

Next: B

A	**3**	**3**	**3**	**4**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
B	**2**	**W**	**W**	**W**	**W**	**3**	**3**	**4**	**4**	**5**	**0**	**0**	**0**	**0**
C	**2**	**2**	**w**	**w**	**W**	**W**	**3**	**3**	**w**	**W**	**w**	**4**	**5**	**0**

Scheduling

Task	length	Arrival Time
0	85	0
1	30	10
2	35	15
3	20	80
4	50	85

FIFO AVG:110

Task	Completion Time	Respond Time
0	85	85
1	115	105
2	150	135
3	170	90
4	220	135

RR AVG: 104

Task	Completion Time	Respond Time
0	220	135
1	80	70
2	135	120
3	145	65
4	215	130

SJF AVG:79

Task	Completion Time	Respond Time
0	220	220
1	40	30
2	75	60
3	100	20
4	150	65

What is “internal fragmentation” in paged memory architecture? What happens when page frames size is too big or too small?

If a process does not use all of the memory inside a frame, it creates “internal fragmentation”. It happens when page frame is too big.

When the operating system reuses memory, it must first zero out the contents of the memory or disk. Why?

Private data from one application could inadvertently leak into another, potentially malicious application.

Benefits of paging:

Efficient memory allocation

Efficient lookup

Efficient reverse lookup

Page-granularity protection and sharing

First fit, best fit, Worst fit:

First fit:

Assign the first large enough space, and the search can start from the beginning or from the last first end. You can stop once you find enough free space.

Best fit:

Allocate the smallest sufficiently large hole and the entire list should be looked up unless the list is sorted by size. This method can produce minimal residual holes

Worst fit:

Assign the largest hole, and again, the entire list should be looked up, unless the list is sorted by size. This method can produce a maximum residual hole, which may be more suitable than the smaller residual hole produced by optimal adaptation.

MIN

Frame	3	2	4	3	4	2	2	3	4	5	6	7	7	6	5	4	5
0	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5	5	5
1		2	2	2	2	2	2	2	2	2	6	6	6	6	6	6	6
2			4	4	4	4	4	4	4	4	4	7	7	7	7	4	4

FIFO

Frame	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
0	7	7	7	2	2	2	2	4	4	4	0	0	0	0	0	0	0	7	7	7
1		0	0	0	0	3	3	3	2	2	2	2	2	1	1	1	1	1	0	0
2			1	1	1	1	0	0	0	3	3	3	3	3	2	2	2	2	2	1

LRU

Frame	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
0	7	7	7	2	2	2	2	4	4	4	0	0	0	1	1	1	1	1	1	1
1		0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0
2			1	1	1	3	3	3	2	2	2	2	2	2	2	2	2	7	7	7

Advantages to Memory-mapped Files

Transparency

Zero copy I/O

Efficient with large file I/O

Disk Access Time:

Disk Access Time = seek time + rotation time + transfer time

Spindle speed 7200 RPM

AVG seek time   10.5/12 ms

Track to track seek time   1ms

Transfer rate (surface to buffer)   54-128 mb/s

Transfer rate (buffer to host)   375mb/s

Buffer memory   16 mb

For 500 sectors(randomly):

Seek time 10.5

Rotation time = 60/7200*1000 /2 = 4.15 ms

Transfer rate = 54 mb/s then for 512b, it will be 512/(54*1024*1024)= 9.04 e-3 ms

Access time = 10.5+4.15+9.04e-3 = 14.66 ms

For 500 requests, the access time will be 14.66*500 = 7.33s

For 500 sectors(sequence):

Transfer rate lower bound: 500*512 * 1/(54*1024*1024) = 4.5ms

Transfer rate upper bound: 500*512*1/(128*1024*1024)= 1.9ms

Access time:

Inner check:

10.5+4.15+1.9 = 16.55

Outer check:

10.5+4.15+4.5 = 19.15

Suppose a variation of FFS includes in each inode 12 direct, 1 indirect, 1 double indirect, 2 triple indirect, and 1 quadruple indirect pointers. Assume 6KB blocks and 6-byte pointers.

What is the largest file that can be accessed via direct pointers only?

number of inode direct: 12, 6 KB blocks, therefore: file size = 12*6 = 72.

What is the maximum file size this index structure can support?

There are 6 kb per block, and 6 bytes for each pointer, therefore, one block has 6kb/6 = 1024 pointers. Then a file can have 12 + 1024 + 1024*1024 + 1024*1024*1024 + 1024*1024*1024*1024 = 1.1005864e+12 blocks. Therefore, the largest file will be

1.1005864e+12 * 6/1024/1024 = 6297606 GB. It is about 6PB

Scans:

Request queue: 53, 98, 183, 37, 122, 14, 124, 65, 67

FCFS (first come first serve)

98, 183, 37,122,14,124,65,67

SSTF (shortest seek time first)

53,65,67,37,14,98,122,124,183

SCAN: (inner to outer, then outer to inner)

53,65,67,98,122,124, 183,37,14

CSCAN:

53,65,67,98,122,124, 183,0(turn around),14,47

RSCAN: (向右扫描，向左的时候不会扫描任何经过的请求直到碰到 0)

53,65,67,98,122,124,183,0,14,37

FAT NTFS FFS

FAT: File Allocation Table, linked list, still widely used in devices like camera. Developed by Microsoft.

Pro:

easy to find free block

easy to append to a file

easy to delete a file

Cons:

Small file access very slow

Random access is very slow

fragmentation

FFS: Fast File System. Tree-based multilevel index, developed by Unix

Pro: Efficient storage for both small and large files

Locality for both small and large files

Locality for metadata and data

Cons:

Inefficient for tiny files

Inefficient encoding when file is mostly contiguous on disk

Need to reserve 10-20 of free space to prevent fragmentation

NTFS: Microsoft New Technology File System, tree-based index. More fixable than FFS fixed tree.