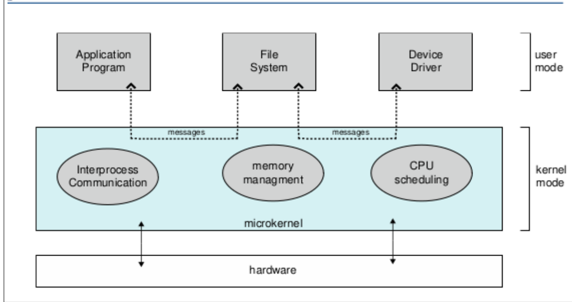
Abstract data type (ADT): internal vars only accessible by code within procedure.
Amdahl's Law: identifies performance gains from adding additional cores to an application that has both serial and parallel compnents. S serial portion, N processing cores. Speedup $\leq 1/(S+ (1-S)/N)$. As $N \to$ infinity, speedup approaches 1/S.
Bootstrap: loaded at boot, stored in ROM (firmware). Loads kernel. Assumes equally divisible parallel code.
Buffer: unbounded: no limit on size of buffer. Consumer wait if empty, producer never waits. Bounded: fixed buffer size. Consumer wait if empty, producer wait if full.
Computer system 4 parts: CPU, memory, disk, I/O devices.
Concurrent: Supports more than one task making progress. Processes start at different time but takes turns running and switching processes.
Context: represented by PCB. System save state of current process and load saved state for new process through this. Made up of register set, stack, private storage area is context of thread.
Deadlock: two or more processes are waiting indefinitely for event that those processes only can cause. Allow at most n-1 processes simultaneously.
Dual-mode: operation allows OS to protect itself and other components. User and Kernel mode. Mode bit from hardware. System call changes mode to kernel, then resets to user mode. Privileged code only executable in kernel mode.
HDD: disk surface divided into tracks, which are subdivided into sectors. Disk controller determines logical interaction between device and computer.
I/O devices and CPU can execute concurrently. Each device controller has local buffer. CPU moves data from local to main memory. I/O is from device to local buffer of controller. Controller informs CPU with interrupt.
Interrupt vector: contains addresses of all service routines for interrupt transfers.
Interprocess communication (IPC): Shared memory and Message passing. Direct: send(p,msg), receive(q,msg). Indirect: send(A,msg), receive(A,msg).
Loopback: special address to refer to system on which process is running.
Long-term scheduler: select which process should be executed next and allocates CPU. Invoked frequently.
Main memory: only large storage media that CPU can directly access. Random access, volatile.
Memory process: text section = program code. Data section = global var. Stack = temp data. Heap = contain memory dynamically allocated during run time.
Monitor: Only one process may be active within monitor at a time. Need additional sync mechanisms such as *condition* vars.
Multiprocessor: parallel systems, tightly-coupled system. Advantages are increase throughput, economy of scale, and increased reliability. Two kinds: Asymmetric and symmetric.
Multithreading Models: Many-to-One: one thread blocking causes all block. Many user threads mapped to single kernel thread. One-to-One: Each user thread maps to kernel thread. Creating user thread creates kernel thread. Number of threads per process restricted for overhead. Many-to-Many: allows many user threads to map to many kernel threads. Allows system to reate sufficient kernel threads Two-level: similar to many-to-many, except allows user thread to bound to kernel thread.
OS is: a resource allocator; a control program; interrupt driven.
Operating services: file-system manipulation, communications, error detection, resource allocation, accounting, protection and security.
Readers-Writers Problem: Problem: allow multiple readers to read at same time while only one writer can access shared data at same time. Solved by reader-writer locks.
Parallelism: Can perform more than one task simultaneously. start at same time on different cores.
Peterson's Solution: Two process share variables int turn: indicate who is in critical section, bool flag[2]: indicate if process is ready to enter critical section. Assumes vars are atomic (cannot be interrupted).
Preemptive: allows preemption of process when running in kernel. Non-preemptive runs until exit kernel mode, blocks, or yields CPU (free of race conditions in kernel mode). Is more responsive, more suitable for real time programming.
Priority Inversion: Scheduling problem when lower-priority process holds lock needed by higher priority process. Solved via priority-inheritance protocol.
Program counter: specifies location of next instruction to execute.
Remote Procedure Calls (RPC): abstracts procedure calls between processes on networked systems.
Secondary storage: extension of main memory providing non-volative storage.
Semaphore: uses wait() and signal(). More sophisticated for sync. Counting semaphore: integer value can range over unrestricted domain. Binary semaphore: integer value can range 0 and 1 (same as mutex).
Short-term scheduler: selects which processes should be brought into ready queue (loaded into memory). Invoked infrequently.
Spinlock: mutex busy waiting. Takes CPU time but avoids context switching. Use on mutli-proc systems. Never be preempted.
Starvation: indefinite blocking.
States: new, running, waiting, ready, terminated.
Stubs: client side proxy for actual procedure for RPC.
Thread: sharing global variables is faster than sharing memory (processes). Pros: responsiveness, availability, resource sharing, economy and speed.
Thread-local storage (TLS): allows each thread to have its own copy of data.
Time-sharing(multitasking): logical extension in which CPU switches jobs so frequently that users can interact with each job with running, creating interactive computing.
Trap or exception: software interrupt.
Zombie: has entry in process table, parent not called wait() but not terminated yet, resources been reallocated by OS.

Threads with posix
```
pthread_create(&tid, 0, function, NULL);
pthread_cancel(tid);
```

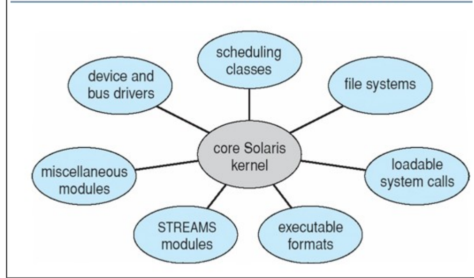
Critical section solution
```
do{     acquire(lock)
                critical section
        release(lock)
                remainder section
} while(true)

acquire(){      while(!available){ //busy wait }
                available = false; }

release(){ available = true; }
```

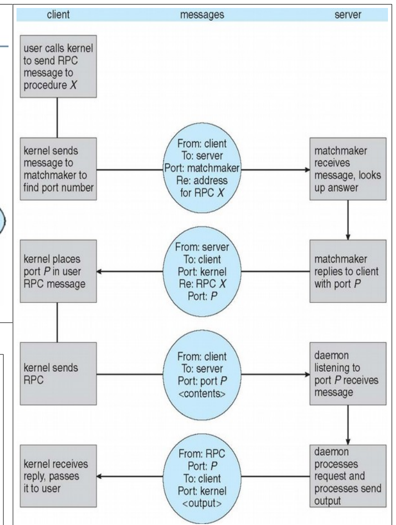
Semaphore
```
P1:     S1; signal(synch);
P2:     wait(synch); S2;

wait(S){        while(S <= 0) { //busy wait }
                --S; }

signal(S) { ++S; }
```

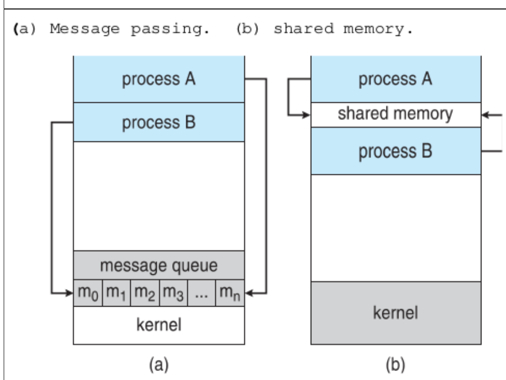
Microkernel System Structure

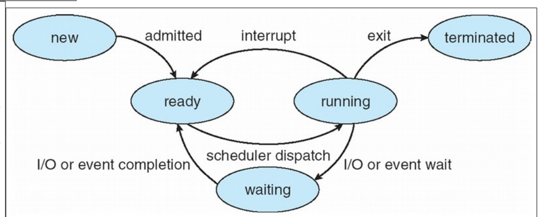


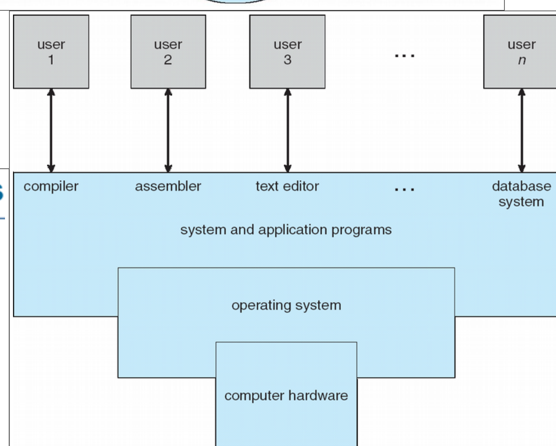
Solaris Modular Approach

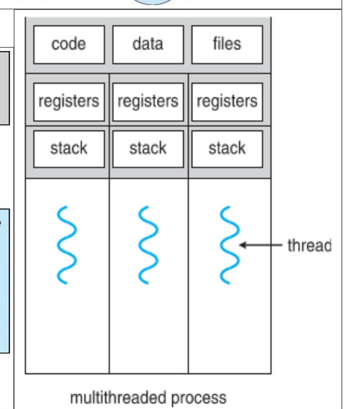




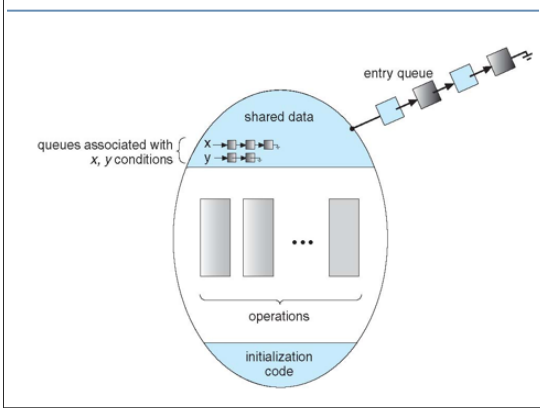








Monitor with Condition Variables



Consumer

```
int val, i = itemCnt;
while(i > 0){
        while(GetIn() == out)
                ;
        val = ReadAtBufIndex(out);
        printf("Consuming Item %d 

        out = (out + 1) % bufSize;
        --i;
        SetOut(out);
}
```

Producer

```
in = GetIn();
out = GetOut();
i = itemCnt;
while(i > 0){
        while(((in + 1) % bufSize) == GetOut())
                ;
        int val = GetRand(0, 5000);
        WriteAtBufIndex(in, val);
        printf("Producing Item %d with value %d
        in = (in + 1) % bufSize;
        --i;
        SetIn(in);
}
```

Solution to Dining Philosophers

Each philosopher *i* invokes the operations **pickup()** and **putdown()** in the following sequence:

```
DiningPhilosophers.pickup(i);

    EAT

DiningPhilosophers.putdown(i);
```

Is a deadlock possible? Is starvation possible?

- No deadlock, but starvation is possible