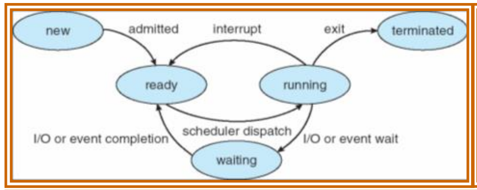
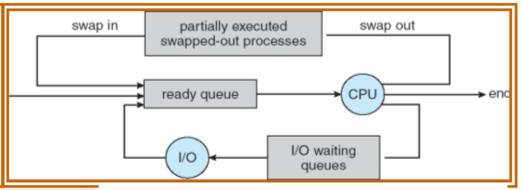
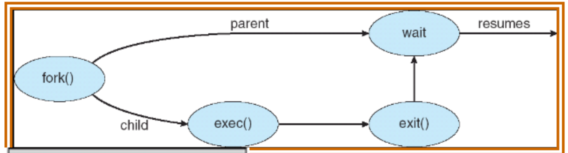
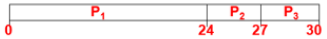




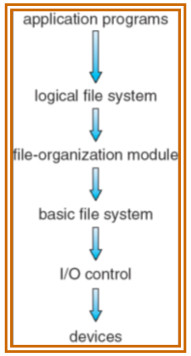
	Max	Allocation	Need	Available	work	分配前	释放后	Finish
	A B C	A B C	A B C	A B C	A B C	A B C	A B C	
P0	7 5 3	0 1 0	7 4 3	3 3 2	3 3 2	④ 7 4 5	7 5 5	
P1	3 2 2	2 0 0	1 2 2			① 3 3 2	5 3 2	
P2	9 0 2	3 0 2	6 0 0			⑤ 7 5 5	10 5 7	
P3	2 2 2	2 1 1	0 1 1			② 5 3 2	7 4 3	
P4	4 3 3	0 0 2	4 3 1			③ 7 4 3	7 4 5	

The system is in a safe state since the sequence < P_1, P_3, P_4, P_0, P_2> satisfies safety criteria.

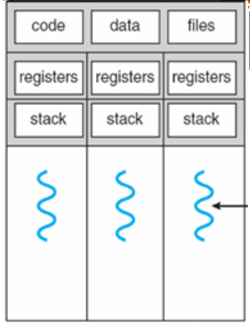


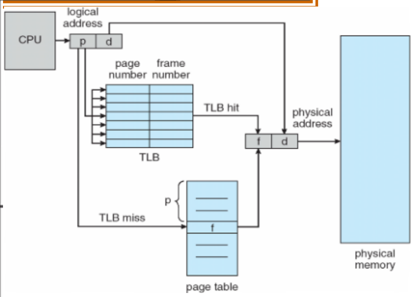


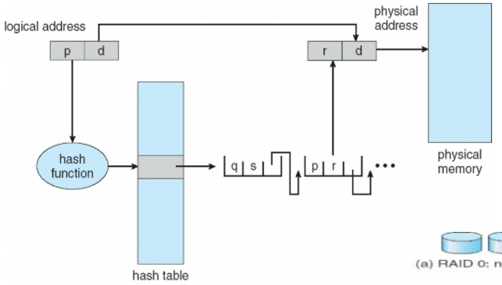
- Waiting time for $P1 = 0$; $P2 = 24$; $P3 = 27$
- Turnaround time for $P1 = 24$; $P2 = 27$; $P3 = 30$
- Average waiting time: $(0 + 24 + 27)/3 = 17$
- Average Turnaround time: $(24 + 27 + 30)/3 = 27$

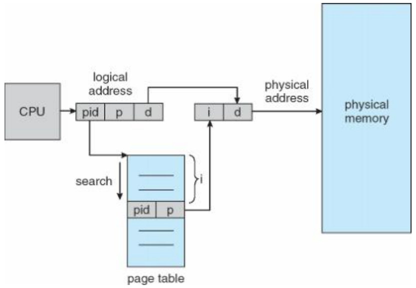


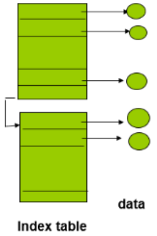
- file permissions
- file dates (create, access, write)
- file owner, group, ACL
- file size
- file data blocks or pointers to file data blocks

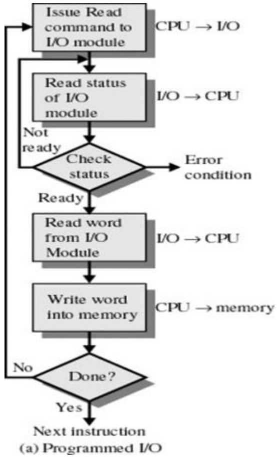




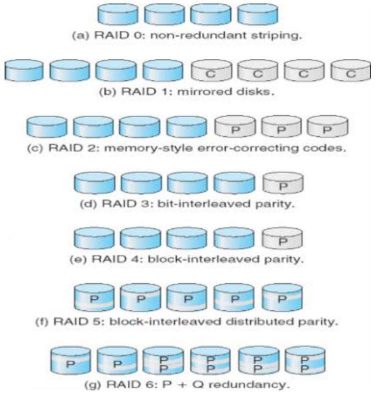








(a) Programmed I/O



(a) RAID 0: non-redundant striping.

(b) RAID 1: mirrored disks.

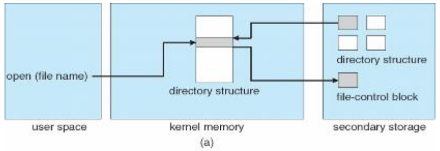
(c) RAID 2: memory-style error-correcting codes.

(d) RAID 3: bit-interleaved parity.

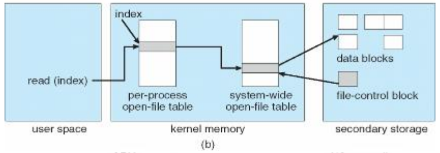
(e) RAID 4: block-interleaved parity.

(f) RAID 5: block-interleaved distributed parity.

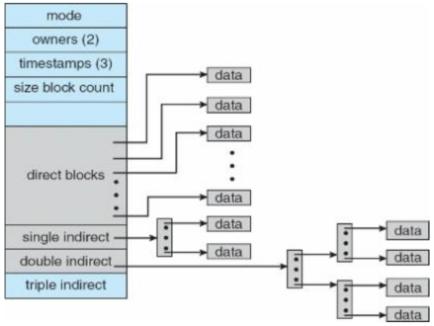
(g) RAID 6: P + Q redundancy.

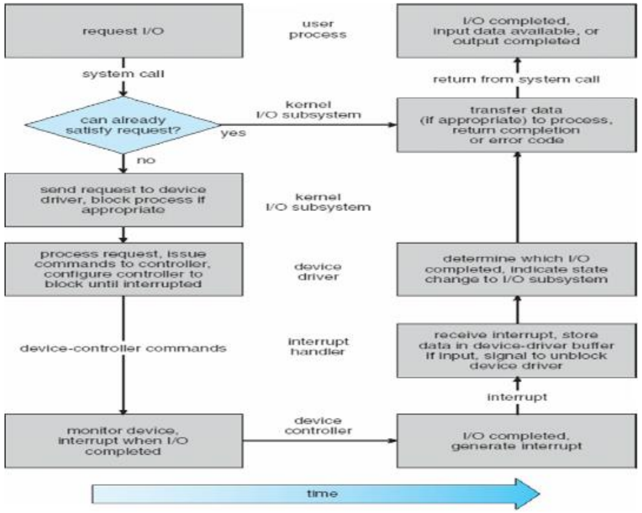


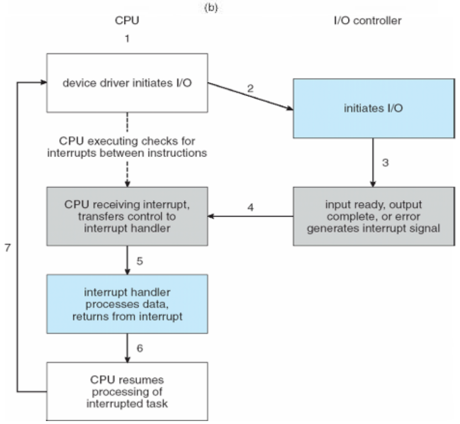
(a)

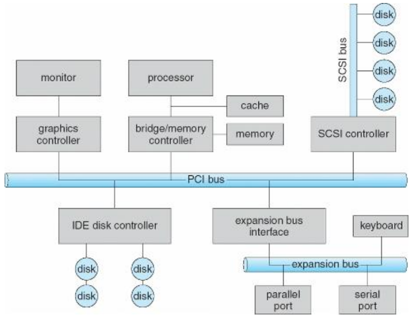


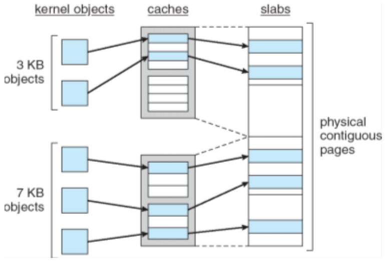
(b)




request I/O
user process
I/O completed, input data available, or output completed
system call
return from system call
can already satisfy request?
yes
no
kernel I/O subsystem
transfer data (if appropriate) to process, return completion or error code
send request to device driver, block process if appropriate
kernel I/O subsystem
process request, issue commands to controller, configure controller to block until interrupted
device driver
determine which I/O completed, indicate state change to I/O subsystem
device-controller commands
interrupt handler
receive interrupt, store data in device-driver buffer if input, signal to unblock device driver
interrupt
monitor device, interrupt when I/O completed
device controller
I/O completed, generate interrupt
time


CPU
I/O controller
1
device driver initiates I/O
2
initiates I/O
CPU executing checks for interrupts between instructions
3
CPU receiving interrupt, transfers control to interrupt handler
4
input ready, output complete, or error generates interrupt signal
5
interrupt handler processes data, returns from interrupt
6
CPU resumes processing of interrupted task
7





```
1-P0: do {
        while (turn != 0) ;
        critical section
        turn = 1;
        remainder section
} while (1);

1-P1: do {
        while (turn != 1) ;
        critical section
        turn = 0;
        remainder section
} while (1);

2- P0: do {
        flag[0] = true;
        while (flag[1]) ;
        critical section
        flag [0] = false;
        remainder section
} while (1);

2- P1: do {
        flag[1] = true;
        while (flag[0]) ;
        critical section
        flag [1] = false;
        remainder section
} while (1);

2.1- P0: do {
        while (flag[1]) ;
        flag[0] = true;
        critical section
        flag [0] = false;
        remainder section
} while (1);

3- Pi: do {
        flag[i]= true;
        turn = j;
        while (flag[j] and turn = j) ;
        critical section
        flag[i] = false;
        remainder section
} while (1);

Swap：while(1) {
        key = TRUE;
        while (key == TRUE) Swap(&lock,&key);
        critical section
        lock = false;
        remainder section
}

wait(S) { while (S <= 0); S--; }
signal(S) { S++; }

do {
        choosing[i] = true;
        number[i] = max(number[0], number[1], …, number [n – 1])+1;
        choosing[i] = false;
        for (j = 0; j < n; j++) {
                //如果 j 在获取排队登记号则等待
                while (choosing[j]) ;
                //如果 j 的序号比 i 小则等待
                while ((number[j] != 0) && (number[j],j) < (number[i],i)) ;
        }
        critical section
        number[i] = 0;
        remainder section
} while (1);
```

使用 TestAndSet 的互斥实现：

```
boolean TestAndSet(boolean &target) {
        boolean rv = target;
        target = true;
        return rv;
}
while(1) {
        while (TestAndSet(lock)) ;
        critical section
        lock = false;
        remainder section
};

while(1) {
        waiting[i]=true;
        key= true;
        while ( waiting[i] && key )    key=TestAndSet(lock);
        waiting[i]=false;
        critical section;
        j= (i+1) % n ;
        while ( (j != i) && !waiting[j] )    j= (j+1) % n ;
        if (j == i)    lock = false ;
        else waiting[j] = false ;
        remainder section ;
}

wait(semaphore*S) {
        S->value--;
        if (S->value<0) {
                add this process to S->list;    block();
        }
}
signal(semaphore *S) {
        S->value++;
        if (S->value <= 0) {
                remove a process P from S->list;
                wakeup(P);
        }
}
```

生产者：

```
do {
        wait(empty); wait(mutex);
        ...
        signal(mutex); signal(full);
} while (1)
```

消费者：

```
do {
        wait(full); wait(mutex);
        ...
        signal(mutex); signal(empty);
} while (1)
```

作家：

```
do {
        wait(rw_mutex);
        ...
        signal(rw_mutex);
} while (1)
```

读者：

```
do {
        wait(mutex);
        read_count++;
        if (read_count == 1) wait(rw_mutex);
        signal(mutex);
        ...
        wait(mutex);
        read_count--;
        if (read_count == 0) signal(rw_mutex);
        signal(mutex);
} while (1)

do {
        wait(chopstick[i]);
        wait(chopstick[(i+1)%5]);
        ...
        wait(chopstick[i]);
        wait(chopstick[(i+1)%5]);
        ...
} while (1)
```