作业答案参考 （1~6）

文件线上更新地址：

https://level-rosemary-521.notion.site/answer-1a8b8b01a07741c88a9cc5d7a023be54#8e8fbe69d9fa4221b835d76acaa1aed0

HW 1

1.

> 操作系统的三个主要作用是什么?

便于使用（接口）、资源管理、控制程序的运行（或者回答：机器扩充，虚拟机也行）

2.

> 从开机开始，简要说明计算机系统是如何一步一步启动操作系统的。

- Power-on→Bootstrap: BIOS→BootLoader: GRUB→OS: Linux
- 打开电源或重启时，位于固件的引导程序（bootstrap program）初始化系统的所有部分，
- 定位操作系统内核并把它装入内存，然后操作系统开始执行第一个进程。

3.

> 按照ppt中的helloworld源代码，生成可执行文件，请给出该文件的入口地址，并给出main函数的地址，这两个地址是同一个吗？为什么?

不是同一个地址。因为，要为进入main函数准备上下文，所以在main进行一些准备。

4.

> 根据你的理解，从命令行运行helloworld程序到该程序运行结束，需要哪些功能的支持?

编码，编译，链接，装载等

5.

> 名词解释：脱机I/O（目的、方法）

目的：解决人机矛盾和CPU与I/O设备之间速度不匹配的矛盾；方法：利用低速的外围机进行IO，例如：纸带（卡片）→磁带（磁盘）

6.

> 名词解释：2阶段调度（2-phrase of scheduling）（哪2阶段？）

作业和进程，前者决定哪个（些）作业进入系统运行，后者决定为哪个进程分配CPU运行

HW 2

1.

> 三种最基本的操作系统类型是什么?

我们平常所使用的操作系统是什么类型的？批处理、分时、实时；三种基本类型的混合。

2.

> Which of the following instructions should be privileged?
>
> a. Set value of timer.
>
> b. Read the clock.
>
> c. Clear memory.
>
> d. Issue a trap instruction.
>
> e. Turn off interrupts.
>
> f. Modify entries in device-status table.

> g. Switch from user to kernel mode. h. Access I/O device.

a c e f h

3.

> 在哪些情况下，我们使用的个人笔记本电脑运行在管理/特权/内核模式？如果用户程序要主动触发进入操作系统内核中执行，据你了解可以如何进行？（提供一种方法即可）

启动过程中、处理中断/异常/系统调用时；可以调用系统调用，或者故意制造一个异常，例如除0错。其他合理的答案也可。

4.

> 用户接口有哪几种？OS提供服务最基本的方式是什么？什么是系统调用号？系统调用参数的传递方法常见的有哪三种？

命令、编程、图形；系统调用；每个系统调用都有一个与之相对应的数字，用户通过指明系统调用号来调用相对应的系统调用，内核使用系统调用号来查找到对用的系统调用实现的入口。寄存器、块或表、栈

5.

> 进程间通信的模型有哪两种？常见的IPC机制包括哪5种？

消息传递和共享内存；信号、信号量、管道、消息队列、套接字

6.

> OS特征有哪些？其中，最基本的特征是什么？并发和并行的关系是什么？异步性具体指什么？

并发、共享、虚拟、异步；并发和共享；并发定义，并行定义，并发包含并行（并行是并发的一种特殊情况）；程序运行的进度不可预知

7.

> 请列举7种不同的OS体系结构？根据你的理解，Linux操作系统采用了什么样的体系结构？

简单、宏、模块、层次、微、混合、外核；Linux以单一大内核（即宏内核）为主，在此基础上有层次化和模块化。

8.

> 程序并发执行的特征和条件分别是什么?

间断性、失去封闭性、不可再现性；Bernstein条件：R (p1) ∩W(p2) ∪ R (p2) ∩W(p1) ∪W(p1) ∩W(p2) = ∅

9.

> 进程的5大特征是什么？其中，最基本的特征是哪个，其具体含义是什么？进程的独立性具体指什么?

动态、并发、独立、异步、结构；动态性；具有生命期、“…”；进程是…基本单位

10.

> 谈谈你对进程地址空间中堆和栈的理解。

堆栈是进程地址空间的两个重要组成部分。堆用于动态空间管理，用户可以根据需要使用malloc/free接口来动态的申请/释放存储空间。堆内的管理算法由C库提供。操作系统为进程管理整个堆空间，包括初始化堆空间、提供相关接口适时扩张或缩小堆空间等。栈一般位于用户态地址空间的高端，向低地址端扩展，主要用于局部变量的分配和函数调用链的保存。其他合理的答案也可。

HW 3

1.

> 在下面的程序中，什么情况下会执行到“LINE J”行（即划横线语句）？

```c
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
pid_t pid;

    /* fork a child process */
    pid = fork();

    if (pid < 0) { /* error occurred */
       fprintf(stderr, "Fork Failed");
       return 1;
    }
    else if (pid == 0) { /* child process */
       execlp("/bin/ls","ls",NULL);
       printf("LINE J");
    }
    else { /* parent process */
       /* parent will wait for the child to complete */
       wait(NULL);
       printf("Child Complete");
    }

    return 0;
}
```

当子进程创建成功后，调用execlp()加载/bin/ls程序的代码和静态数据，若调用成功，则永远不会执行printf(“line J”)；若调用失败（如不存在/bin/ls这样的可执行程序），才会执行printf(“line J”)

2.

> 什么是调度的饥饿现象？在我们介绍的调度算法中，哪几个调度算法可能会导致产生饥饿现象？择其一举例说明。

调度的饥饿现象指的是某些进程由于优先级过低或其他原因长期得不到CPU资源，导致其无法运行。可能导致饥饿现象的调度算法包括：

- 优先级调度（Priority Scheduling）
- 最短作业优先（SJF）

例如，优先级调度算法中，如果总是有高优先级的进程到达，低优先级的进程可能一直无法获得CPU时间，从而产生饥饿现象。

3.

The following processes are being scheduled using a preemptive, priority-based, round-robin scheduling algorithm.

Process	Priority	Burst	Arrival
P_1	8	15	0
P_2	3	20	0
P_3	4	20	20
P_4	4	20	25
P_5	5	5	45
P_6	5	15	55

Each process is assigned a numerical priority, with a higher number indicating a higher relative priority. The scheduler will execute the highest-priority process. For processes with the same priority, a round-robin scheduler will be used with a time quantum of 10 units. If a process is preempted by a higher-priority process, the preempted process is placed at the end of the queue.

a. Show the scheduling order of the processes using a Gantt chart.

b. What is the turnaround time for each process?

c. What is the waiting time for each process?

a.

调度过程如下：

时间段	当前执行任务	队列任务
0-10	P1	P2
10-15	P1	P2

时间段	当前执行任务	队列任务
15-20	P2	
20-25	P3	P2
25-30	P3	P4, P2
30-40	P4	P3, P2
40-45	P3	P4, P2
45-50	P5	P4, P3, P2
50-55	P4	P3, P2
55-70	P6	P3, P4,P2
70-75	P3	P4, P2
75-80	P4	P2
80-95	P2	

```
| P1  | P2  | P3  | P4  | P3  | P5  | P4  | P6  | P3 | P4  | P2  |

 0    15   20    30    40    45    50    55    70    75   80     95
```

b.

turnaround 时间为完成时间 - 到达时间

进程	到达时间	完成时间	周转时间
P1	0	15	15
P2	0	95	95
P3	20	75	55
P4	25	80	55
P5	45	50	5
P6	55	70	15

c.

waiting 时间为周转时间 - 执行时间

进程	执行时间	周转时间	等待时间
P1	15	15	0
P2	20	95	75
P3	20	55	35
P4	20	55	35
P5	5	5	0
P6	15	15	0

4.

> 现有两个进程P1和P2，已知，p1=d1=50，t1=25，p2=d2=75，t2=30。
> a) 这两个进程是否能通过RM调度算法来调度？请分别使用CPU利用率和作图法（甘特图）来解释。
>
> b) 试用EDF调度算法来调度这两个进程。（给出甘特图）

a.

- CPU利用率：

$$\text{CPU利用率} = \frac{t1}{p1} + \frac{t2}{p2} = \frac{25}{50} + \frac{30}{75} = 0.5 + 0.4 = 0.9$$

  因为利用率大于 0.83，所以这两个进程不可以通过RM调度算法调度。
- 甘特图法：
  创建甘特图，从时间0开始，P1执行25个时间单位，然后P2执行25个时间单位，然后再次执行P1。

```
|P1 |P2 |P1 |

0   25  50  75
```

b.

- EDF调度算法：
  在每个调度点选择相对截止时间最早的任务。
  i. P1的截止时间为50，P2为75，首先调度P1。
  ii. P1运行25个时间单位到达25。
  iii. 接着P2运行30个时间单位到达55。
  iv. 重复以上过程。

```
|P1 |P2 |P1 |

0   25  55  80
```

5.

> 操作系统可能要支持多种不同的进程在同一个系统中运行。例如IO密集的、CPU密集的、实时的，等等。在调度模块中只实现某一种调度算法，是否合适？如果不合适，怎样做合适？

在调度模块中只实现某一种调度算法是不合适的，因为不同类型的进程对资源的需求不同。例如，IO密集型进程需要频繁地进行IO操作，CPU密集型进程需要大量的CPU时间，实时进程则对响应时间有严格要求。因此，应该采用多种调度算法的组合来满足不同类型进程的需求，例如：

- 为实时进程使用EDF或RM调度算法。
- 为普通进程使用多级反馈队列调度算法。
- 为IO密集型进程和CPU密集型进程分别优化调度策略。

通过这样的组合，可以有效地提高系统的整体性能和响应速度。

HW 4

1.

> 下面的程序采用Pthreads API。该程序的LINE C和LINE P的输出分别是什么？

```
#include <pthread.h>
#include <stdio.h>

int value = 0;
void *runner(void *param); /* the thread */

int main(int argc, char *argv[])
{
pid_t pid;
pthread_t tid;
pthread_attr_t attr;

   pid = fork();

   if (pid == 0) { /* child process */
      pthread_attr_init(&attr);
      pthread_create(&tid,&attr,runner,NULL);
      pthread_join(tid,NULL);
      printf("CHILD: value = %d",value); /* LINE C */
   }
   else if (pid > 0) { /* parent process */
      wait(NULL);
      printf("PARENT: value = %d",value); /* LINE P */
   }
}

void *runner(void *param) {
   value = 5;
   pthread_exit(0);
}
```

- line c: child: value = 5
- line p parent: value = 0

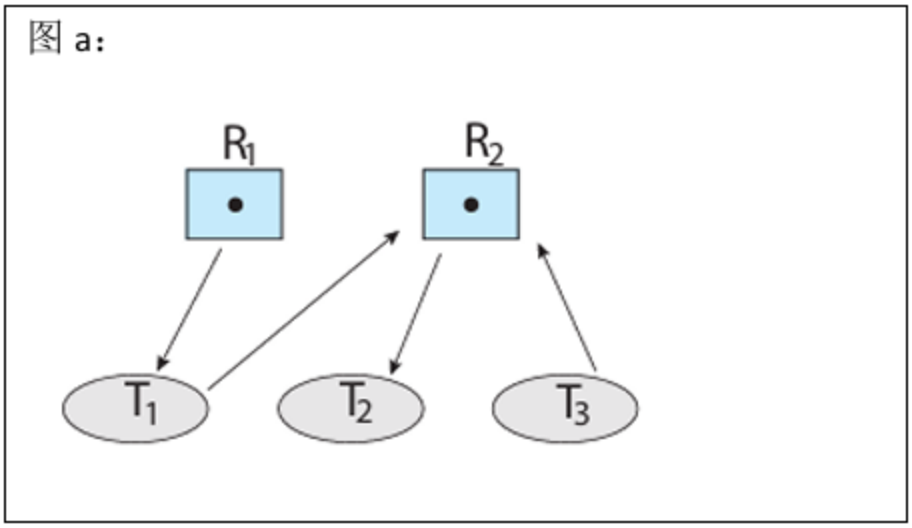
2.

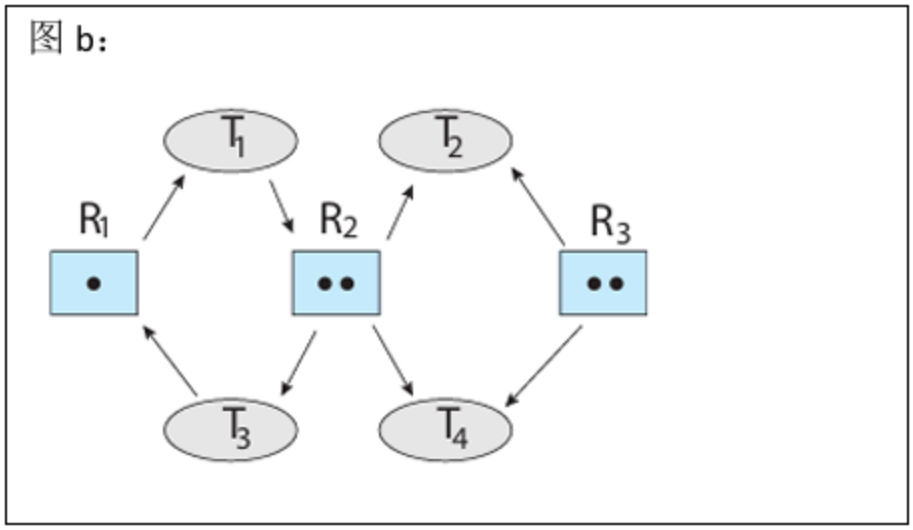
> PPT 中给出的读者-写者问题的解决方案，是否会导致饥饿现象？如果会，请列举一种可能会产生饥饿现象的场景。

会导致饥饿现象。读者一直存在，写者会一直等待。

3.

> 下面的两个图中，哪个(些)处于死锁状态？如果处于死锁状态，请列出处于死锁的线程和资源形成的环。如果不是处于死锁状态，请列出可能顺利完成执行的线程顺序。





(图 b 中 R2->T2 改成 T2->R2)

1. 无环 无死锁，T2 -> T3 -> T1
2. 有环 无死锁，T4 -> T2 -> T1 -> T3

4.

> PPT 第36页，两个问题中选择一个回答。起始状态为完成(1,0,2)的分配之后。

初始状态

	Allocation	Need	Available
	ABC	ABC	ABC
P0	0 1 0	7 4 3	2 3 0
P1	3 0 2	1 2 2	

	Allocation	Need	Available
P2	3 0 1	6 0 0	
P3	2 1 1	0 1 1	
P4	0 0 2	4 3 1	

1. Can request for (3,3,0) by P4 be granted?
   - R1 分量不足，无法分配。
2. Can request for (0,2,0) by P0 be granted?
   - 试分配 (0,2,0) 给 P0，得

	Allocation	Need	Available
	ABC	ABC	ABC
P0	0 3 0	7 2 3	2 1 0
P1	3 0 2	1 2 2	
P2	3 0 1	6 0 0	
P3	2 1 1	0 1 1	
P4	0 0 2	4 3 1	

此时 Available 不满足任一任务需求，故不安全。

HW 5

1.

> 什么是内部碎片和外部碎片？两者的差异在哪里？

- 内部碎片
  - 指的是在内存分配过程中，由于内存块（通常是固定大小的内存块或页）被分配给一个进程，但该进程所需的内存比分配的块小，导致剩余的内存空间无法被其他进程使用，从而造成的浪费。
  - 特征：
    - 内存块内部的未使用空间
    - 主要发生在固定分区分配或分页系统中。
- 外部碎片

- 指的是在内存分配和释放过程中，由于内存空间被分配和释放的不连续，导致内存中虽然有足够的空闲内存空间，但这些空闲空间是分散的，无法满足进程的大块连续内存请求，从而造成的浪费。
- 特征：
  - 内存块之间的未使用空间。
  - 主要发生在动态分区分配系统中。

2.

> 假设按空闲链表的顺序从前到后有下列6个分区：
>
分区 1	分区 2	分区 3	分区 4	分区 5	分区 6
100MB	170MB	40MB	205MB	300MB	185MB
>
> 现在分别采用first-fit、best-fit 和 worst-fit 算法依次为 6 个进程分配内存，这 6 个进程请求的size如下：
>
P1	P2	P3	P4	P5	P6
200MB	15MB	185MB	75MB	175MB	80MB
>
> 请作图说明这三种算法下的分配情况，并回答：哪种算法会导致哪个请求得不到满足？（假设最小分区大小为1MB）

First-Fit 分配过程

1. **P1 200MB** 分配到分区4（205MB）：
   - 新分区列表：100MB, 170MB, 40MB, 5MB, 300MB, 185MB
2. **P2 15MB** 分配到分区1（100MB）：
   - 新分区列表：85MB, 170MB, 40MB, 5MB, 300MB, 185MB
3. **P3 185MB** 分配到分区5（185MB）：
   - 新分区列表：85MB, 170MB, 40MB, 5MB, 115MB, 185MB
4. **P4 75MB** 分配到分区1（85MB）：
   - 新分区列表：10MB, 170MB, 40MB, 5MB, 115MB, 185MB
5. **P5 175MB** 分配到分区6（185MB）：
   - 剩余分区：10MB, 170MB, 40MB, 5MB, 115MB, 10MB
6. **P6 80MB** 分配到分区2（170MB）：
   - 新分区列表：10MB, 90MB, 40MB, 5MB, 115MB, 10MB

Best-Fit 分配过程

1. **P1 200MB** 分配到分区4（205MB）：
   - 新分区列表：100MB, 170MB, 40MB, 5MB, 300MB, 185MB
2. **P2 15MB** 分配到分区3（40MB）：
   - 新分区列表：100MB, 170MB, 25MB, 5MB, 300MB, 185MB
3. **P3 185MB** 分配到分区6（185MB）：
   - 新分区列表：100MB, 170MB, 25MB, 5MB, 300MB, 0MB
4. **P4 75MB** 分配到分区1（100MB）：
   - 新分区列表：25MB, 170MB, 25MB, 5MB, 300MB, 0MB
5. **P5 175MB** 分配到分区5（185MB）：
   - 剩余分区：25MB, 170MB, 25MB, 5MB, 125MB, 0MB
6. **P6 80MB** 分配到分区5（125MB）：
   - 新分区列表：25MB, 170MB, 25MB, 5MB, 45MB, 0MB

Worst-Fit 分配过程

1. **P1 200MB** 分配到分区5（300MB）：
   - 新分区列表：100MB, 170MB, 40MB, 205MB, 100MB, 185MB
2. **P2 15MB** 分配到分区4（205MB）：
   - 新分区列表：100MB, 170MB, 40MB, 190MB, 100MB, 185MB
3. **P3 185MB** 分配到分区4（190MB）：
   - 新分区列表：100MB, 170MB, 40MB, 5MB, 100MB, 185MB
4. **P4 75MB** 分配到分区6（185MB）：
   - 新分区列表：100MB, 170MB, 40MB, 5MB, 100MB, 110MB
5. **P5 175MB** 无法分配:
   - 剩余分区：100MB, 170MB, 40MB, 5MB, 100MB, 110MB
6. **P6 80MB** 分配到分区2（170MB）：
   - 新分区列表：100MB, 90MB, 40MB, 35MB, 100MB, 0MB

Worst-Fit 会导致 P5 分配不了。

3.

> 假设采用1级页表，且页表存于内存中。
>
> a) 假设访存一次耗时50ns，请问访问某逻辑地址中的内容，需要多长时间?
>
> b) 假设使用TLB，且 TLB访问时间2ns，TLB的命中率为75%，则有效访问时间多长?

a.

在使用1级页表的情况下，访问一个逻辑地址中的内容需要两次内存访问：

一次是读取页表项（存储在内存中）。
一次是实际的内存访问。
因此，访存时间 T 可以计算为：

$$T = 2 \times 50ns = 100ns$$

b.

在使用TLB的情况下，有两种情况需要考虑：TLB命中和TLB未命中。

- TLB命中：
  - TLB访问时间 = 2ns
  - 内存访问时间 = 50ns
  - 总时间 = 2ns + 50ns = 52ns
- TLB未命中：
  - TLB访问时间 = 2ns
  - 页表访问时间 = 50ns
  - 内存访问时间 = 50ns
  - 总时间 = 2ns + 50ns + 50ns = 102ns

TLB命中率为75%，因此TLB未命中率为25%。有效访问时间可以用加权平均法计算：

$$EAT = 0.75 \times 52ns + 0.25 \times 102ns = 39ns + 25.5ns = 64.5ns$$

4.

> 假设x86下：
>
> a) 若当前为实模式，CS寄存器中的值为0x07C0，IP寄存器中的值为0x28，则实际要访问的指令物理地址为多少?
>
> b) 若当前为保护模式，并且GDT表中第2、3项描述符中的基地址都是0x00000000，若CS寄存器中的值为0x18，而IP寄存器中的值为0x87654321，则实际要访问的指令线性地址为多少?

a.

在实模式下，物理地址由段寄存器（如CS）和偏移量（如IP）组合而成：

- 物理地址 $= CS \times 16 + IP$

- 将给定的 CS 和 IP 代入：
- 物理地址 $= 0x07C0 \times 16 + 0x28 = 0x07C28$

b.

在保护模式下，段寄存器（如CS）中的值是一个段选择子，指向GDT表中的描述符。描述符包含段基地址、段界限等信息。根据题目中的条件，GDT表中第2、3项描述符的基地址都是0x00000000。

CS寄存器中的值0x18是一个段选择子，其中：

- 索引部分（Index）：0x18 >> 3 = 0x3（选择第3项描述符）
- 表示选择GDT表

第3项描述符的基地址为0x00000000，因此线性地址计算为：

- 线性地址= 基地址 + IP $= 0x00000000 + 0x87654321 = 0x87654321$

HW 6

1.

> 考虑虚存。假设某程序访问某虚拟地址。下列中哪种现象会发生？如果发生，请描述在什么情况下，发生该现象。如果不发生，说明为什么？
>
> a) TLB 缺失，没有缺页
>
> b) TLB缺失，并缺页
>
> c) TLB 命中，没有缺页
>
> d) TLB命中，缺页

a、b、c 可能发生，d 不可能发生

- TLB 缺失，没有缺页
  - 这意味着虚拟地址对应的物理页号不在TLB中。
- TLB缺失，并缺页
  - 如果虚拟地址对应的物理页号既不在TLB中，也不在主存中，就会发生真正的缺页异常。
- TLB 命中，没有缺页
  - 这意味着虚拟地址对应的物理页号在TLB中已经存在。
- TLB命中，缺页
  - 这种情况不会发生。如果TLB命中，说明物理页号已经在TLB中，不会再发生缺页。

2.

> 假设系统具有12位虚拟地址和物理地址，页大小为256字节。后续分配新的物理页框时将按照9，F，D的顺序依次分配。下表是某时刻的页表，其中“-”表示相应的页不在内存中。

页	物理页框
0	0x4
1	0xB
2	0xA
3	-
4	-
5	0x2
6	-
7	0x0
8	0xC
9	0x1

> 请将下面使用16进制表示的逻辑地址转换成物理地址（采用16进制）。若遇到缺页，则分配一个新的物理页框，并使用更新后的页表进行地址转换。（注意：分配物理页框时，按照题中给定的顺序依次分配。）
>
> a) 0x2A1
>
> b) 0x4E6
>
> c) 0x94A
>
> d) 0x316

a) 0x2A1

- 逻辑地址：0x2A1
- 页号：0x2
- 页内偏移量：0xA1

查找页表：

- 页2映射到物理页框0xA。
- 物理地址：页框号0xA * 页大小 + 页内偏移量0xA1 = 0xA00 + 0xA1 = 0xAA1

b) 0x4E6

- 逻辑地址：0x4E6
- 页号：0x4
- 页内偏移量：0xE6

查找页表：

- 页4不在内存中，需要分配新的物理页框。
- 按照给定顺序，分配物理页框9，更新页表。
  - 更新后的页表：
    - 页4：0x9
- 物理地址：页框号0x9 * 页大小 + 页内偏移量0xE6 = 0x900 + 0xE6 = 0x9E6

c) 0x94A

- 逻辑地址：0x94A
- 页号：0x9
- 页内偏移量：0x4A

查找页表：

- 页9映射到物理页框0x1。
- 物理地址：页框号0x1 * 页大小 + 页内偏移量0x4A = 0x100 + 0x4A = 0x14A

d) 0x316

- 逻辑地址：0x316
- 页号：0x3
- 页内偏移量：0x16

查找页表：

- 页3不在内存中，需要分配新的物理页框。
- 按照给定顺序，分配物理页框F，更新页表。
  - 更新后的页表：
    - 页3：0xF
- 物理地址：页框号0xF * 页大小 + 页内偏移量0x16 = 0xF00 + 0x16 = 0xF16

结果

- a) 0x2A1 -> 0xAA1
- b) 0x4E6 -> 0x9E6
- c) 0x94A -> 0x14A
- d) 0x316 -> 0xF16

3.

> 假设进程能使用3个物理页框，请将FIFO、LRU、OPT三种页面置换算法用于下面的访问串：3,1,4,2,5,4,1,3,5,2,0,1,1,0,2,3,4,5,0,1。

FIFO（先进先出）页面置换算法

访问页面	内存状态	页面置换	缺页
3	3		是
1	3, 1		是
4	3, 1, 4		是
2	1, 4, 2	3->2	是
5	4, 2, 5	1->5	是
4	4, 2, 5		否
1	2, 5, 1	4->1	是
3	5, 1, 3	2->3	是
5	5, 1, 3		否
2	1, 3, 2	5->2	是
0	3, 2, 0	1->0	是
1	2, 0, 1	3->1	是
1	2, 0, 1		否
0	2, 0, 1		否
2	2, 0, 1		否

访问页面	内存状态	页面置换	缺页
3	0, 1, 3	2->3	是
4	1, 3, 4	0->4	是
5	3, 4, 5	1->5	是
0	4, 5, 0	3->0	是
1	5, 0, 1	4->1	是

缺页次数：15次

LRU（最近最少使用）页面置换算法

LRU算法是替换最近最少使用的页面。

访问页面	内存状态	页面置换	缺页
3	3		是
1	1, 3		是
4	4, 1, 3		是
2	2, 4, 1	3->2	是
5	5, 2, 4	1->5	是
4	4, 5, 2		否
1	1, 4, 5	2->1	是
3	3, 1, 4	5->3	是
5	5, 3, 1	4→5	是
2	2, 5, 3	1->2	是
0	0, 2, 5	3→0	是
1	1, 0, 2	5->1	是
1	1, 0, 2		否
0	0, 1, 2		否
2	2, 0, 1		否

访问页面	内存状态	页面置换	缺页
3	3, 2, 0	1->3	是
4	4, 3, 2	0->4	是
5	5, 4, 3	2->5	是
0	0, 5, 4	3->0	是
1	1, 0, 5	4->1	是

缺页次数：15次

OPT（最优）页面置换算法

OPT算法是替换将来最长时间不再被使用的页面。

访问页面	内存状态	页面置换	缺页
3	3		是
1	3, 1		是
4	3, 1, 4		是
2	1, 4, 2	3->2	是
5	1, 4, 5	2→5	是
4	1, 4, 5		否
1	1, 4, 5		否
3	3, 1, 5	4 → 3	是
5	3, 1, 5		否
2	3, 1, 2	5 → 2	是
0	1, 2, 0	3 ->0	是
1	1, 2, 0		否
1	1, 2, 0		否
0	1, 2, 0		否
2	1, 2, 0		否

访问页面	内存状态	页面置换	缺页
3	1, 3, 0	2->3	是
4	1, 0, 4	3->4	是
5	1, 0, 5	4->5	是
0	1, 0, 5		否
1	1, 0, 5		否

缺页次数：11次

4.

> 抖动（thrashing，有时被翻译为颠簸）产生的原因是什么？系统如何检测抖动？当检测到抖动时，系统应当如何处理？

1. 抖动产生的原因：
   - 内存不足：系统的物理内存不能满足所有进程的需求，导致频繁的页面置换
   - 过度多任务：同时运行的进程太多，每个进程都需要内存，超出了物理内存的总量
2. 抖动的检测：
   - 抖动通常通过缺页率来检测。如果缺页率急剧增加，系统的实际效率会大幅下降，这可能是抖动的迹象。
   - CPU利用率降低：虽然系统看似忙碌，但大部分时间在进行页面置换而不是实际的进程执行，导致CPU利用率降低
3. 处理策略：
   - 减少运行进程的数量，提高进程优先级
   - 调整页面替换算法，提高页面大小
   - 通过跟踪每个进程的工作集，确保每个进程在内存中的页数足以满足其正常运行。
   - 根据进程的工作集动态调整分配给每个进程的内存页数。

5.

> 考虑某文件系统。采用组合索引方式，其中包含12个直接索引，其它均为一个。磁盘块的大小为8KB，数据块索引指针需要4字节来存储。请问此文件系统中文件最大多大？

1. 直接索引部分：
   - 每个直接索引指向一个数据块。
   - 直接索引数量：12 个。
   - 每个数据块大小：8 KB。

总数据量：

$$12 \times 8\text{KB} = 96\text{KB}$$

2. 单级间接索引：
   - 一个单级间接索引块可以存储的指针数量：

$$\frac{8\text{KB}}{4\text{字节}} = \frac{8192\text{字节}}{4\text{字节}} = 2048\text{个}$$

   - 每个指针指向一个数据块，每个数据块大小为8 KB。

   总数据量：

$$2048 \times 8\text{KB} = 16\text{MB}$$

3. 双级间接索引：
   - 一个双级间接索引块可以存储2048个指向单级间接索引块的指针。
   - 每个单级间接索引块可以指向2048个数据块。

   总数据量：

$$2048 \times 2048 \times 8\text{KB} = 32\text{GB}$$

4. 三级间接索引：
   - 一个三级间接索引块可以存储2048个指向双级间接索引块的指针。
   - 每个双级间接索引块可以指向2048个单级间接索引块。

   总数据量：

$$2048 \times 2048 \times 2048 \times 8\text{KB} = 64\text{TB}$$

总数据量约为：

$$96\text{KB} + 16\text{MB} + 32\text{GB} + 64\text{TB} \approx 64.03125\text{TB}$$

综上所述，此文件系统中，文件的最大大小约为64.03125 TB。

6.

> 假设磁盘有 5000 个柱面，编号 0-4999。假设磁头的当前位置在 2150，此前的位置在1805，对于下面的柱面请求序列（按请求到达的顺序）：
>
> 2069; 1212; 2296; 2800; 544; 1618; 356; 1523; 4965; 3681
>
> 问：分别采用FCFS、SCAN、C-SCAN、C-LOOK这4 中磁盘调度算法，磁头的总位移分别是多少?

FCFS

- FCFS算法按照请求到达的顺序依次处理。
- 总位移为

$$|2150-2069|+|2069-1212|+|1212-2296|+|2296-2800|+|2800-544|$$

$$+|544-1618|+|1618-356|+|356-1523|+|1523-4965|+|4965-3681|$$

- 总位移 $=81+857+1084+504+2256+1074+1262+1167+3442+1284$
- 总位移 $=13011$

SCAN

- SCAN算法按照一个方向移动，直到磁盘一端，然后反向移动。
- 从当前位置向较远的方向（向大柱面号方向）移动，再返回
- 总位移$=(4999-2150)+(4999-356)=7492$

C-SCAN

- C-SCAN算法在一个方向上移动直到最远的请求，然后快速返回到最小的柱面号。
- 总位移 $=(4999-2150)+(4999-0)+(2069-0)=2849+4999+2069=9917$

C-LOOk

- C-LOOK算法只在一个方向上移动到最远的请求，然后快速返回到最近的请求。
- 总位移 $=(4965-2150)+(4965-356)+(2069-356)=9137$

7.

> 常见的I/O控制方式有哪几种?

- 程序 I/O 控制方式
- 中断驱动 I/O 控制方式
- 直接存储器访问 DMA 控制方式
- I/O 通道控制方式