3SH3 HOMEWORK 3
Bradley Kohler

Question 6.5

Given the following state for the Banker's Algorithm.

6 processes P0 through P5

4 resource types: A (15 instances); B (6 instances)

C (9 instances); D (10 instances)

Snapshot at time T0:

Available			
A	B	C	D
6	3	5	4

	Current allocation				Maximum demand			
Process	A	B	C	D	A	B	C	D
P0	2	0	2	1	9	5	5	5
P1	0	1	1	1	2	2	3	3
P2	4	1	0	2	7	5	4	4
P3	1	0	0	1	3	3	3	2
P4	1	1	0	0	5	2	2	1
P5	1	0	1	1	4	4	4	4

a) Verify that the Available array has been calculated correctly.
b) Calculate the Need matrix.
c) Show that the current state is safe, that is, show a safe sequence of processes. In addition, to the sequence show how the Available (working array) changes as each process terminates.
d) Given the request (3,2,3,3) from Process P5. Should this request be granted? Why or why not?

Answer 6.5

a) Currently allocated to Processes P0-P5:

A	B	C	D
9	3	4	6

Instances of resources:

A	B	C	D
15	6	9	10

Subtraction of the two:

A	B	C	D
6	3	5	4

3SH3 HOMEWORK 3
Bradley Kohler

The Available array has been calculated correctly.

b) The need matrix:

	A	**B**	**C**	**D**
P0	7	5	3	4
P1	2	1	2	2
P2	3	4	4	2
P3	2	3	3	1
P4	4	1	2	1
P5	3	4	3	3

c) P4 Executes:

A	**B**	**C**	**D**
7	4	5	4

P5 Executes:

A	**B**	**C**	**D**
8	4	6	5

P2 Executes:

A	**B**	**C**	**D**
12	5	6	7

P0 Executes:

A	**B**	**C**	**D**
14	5	8	8

P1 Executes:

A	**B**	**C**	**D**
14	6	9	9

P3 Executes:

A	**B**	**C**	**D**
15	6	9	10

All instances of resources are now available.

d) This request should be granted. The available matrix has enough resources for P5 to execute request 3,2,3,3.

3SH3 HOMEWORK 3
Bradley Kohler

Question 6.6

In the code below, three processes are competing for six resources labeled A to F.

a) Using a resource allocation graph (Figures 6.5 and 6.6), show the possibility of a deadlock in this implementation.
b) Modify the order of some of the get requests to prevent the possibility of any deadlock. You cannot move requests across procedures, only change the order inside each procedure. Use a resource allocation graph to justify your answer.

```
void P0()
{
  while (true) {
    get(A);
    get(B);
    get(C);
    // critical region:
    // use A, B, C
    release(A);
    release(B);
    release(C);
  }
}
```

```
void P1()
{
  while (true) {
    get(D);
    get(E);
    get(B);
    // critical region:
    // use D, E, B
    release(D);
    release(E);
    release(B);
  }
}
```

```
void P2()
{
  while (true) {
    get(C);
    get(F);
    get(D);
    // critical region:
    // use C, F, D
    release(C);
    release(F);
    release(D);
  }
}
```

Answer 6.6

a) A deadlock is possible in the case where P2 executes get(C), P0 executes until get(C), P1 executes until get(B) then finally P2 execute until get(D). In this case P0 will be waiting on C, P1 will be waiting on B and P2 will be waiting on D.

For the above scenario:

Available:

A	B	C	D	E	F
0	0	0	0	0	0

Current Allocation:

	A	B	C	D	E	F
P0	1	1	0	0	0	0
P1	0	0	0	1	1	0
P2	0	0	1	0	0	1

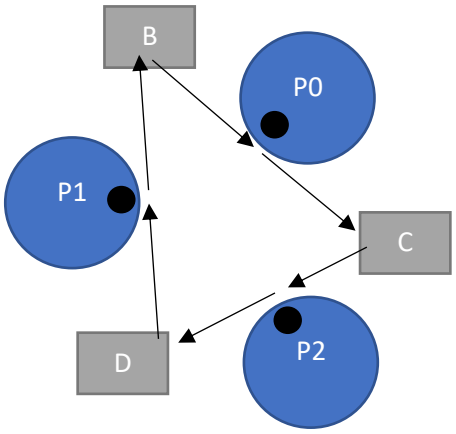
Need Matrix:

	A	B	C	D	E	F

P0	0	0	1	0	0	0
P1	0	1	0	0	0	0
P2	0	0	0	1	0	0

As you can see, no resources are currently available and all processes need at least one resource.

The resource allocation graph of this scenario is as follows:

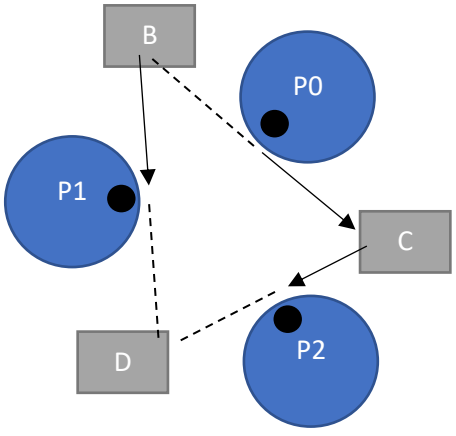


This is a circular wait.

b) One order that would prevent deadlock is:

P0	P1	P2
get(C)	get(B)	get(C)
get(B)	get(D)	get(D)
get(A)	get(E)	get(F)

The resource allocation graph of this scenario is as follows:



Please ignore the dotted lines.

As you can see, at any given time one process will not be interacting with the resources that the other two processes share. This means that deadlock is avoided because a circular wait is not possible.

Question 6.11

Consider a system with a total of 150 units of memory, allocated to three processes as shown:

Process	Max	Hold
1	70	45
2	60	40
3	60	15

Apply the banker's algorithm to determine whether it would be safe to grant each of the following requests. If yes, indicate a sequence of terminations that could be guaranteed possible. If no, show the reduction of the resulting allocation table.

a) A fourth process arrives, with a maximum memory need of 60 and an initial need of 25 units.
b) A fourth process arrives, with a maximum memory need of 60 and an initial need of 35 units.

Answer 6.11

a) The allocation table is as follows:

Process	Max	Hold
1	70	45
2	60	40
3	60	15
4	60	25

The available array is:

Memory
25

The need matrix is:

Process	Memory
1	25
2	20
3	45
4	35

A sequence where all programs execute is [P1, P2, P3, P4].

b) The allocation table is as follows:

3SH3 HOMEWORK 3
Bradley Kohler

Process	Max	Hold
1	70	45
2	60	40
3	60	15
4	60	35

The available array is:

Memory
15

The need matrix is:

Process	Memory
1	25
2	20
3	45
4	25

Not enough memory is available for any of the processes to execute.

Question 6.14

Suppose the following two processes, foo and bar, are executed concurrently and share the semaphore variables S and R (each initialized to 1) and the integer variable x (initialized to 0).

```
void foo( ) {
    do {
      semWait(S);
      semWait(R);
      x++;
      semSignal(S);
      SemSignal(R);
    } while (1);
}
```

```
void bar( ) {
do {
    semWait(R);
    semWait(S);
    x--;
    semSignal(S;
    SemSignal(R);
} while (1);
}
```

a) Can the concurrent execution of these two processes result in one or both being blocked forever? If yes, give an execution sequence in which one or both are blocked forever.
b) Can the concurrent execution of these two processes result in the indefinite postponement of one of them? If yes, give an execution sequence in which one is indefinitely postponed.

Answer 6.14

a) Yes, both processes can be blocked forever. Consider the following scenario:
   i) Process foo() executes semWait(S) and proceeds
   ii) Process bar() executes semWait(R) and proceeds
   iii) Process foo() executes semWait(R) and waits

iv) Process bar() executes semWait(S) and waits

b) No. There is no scenario where one process is waiting in the loop while the other finishes execution of a single loop cycle.

Question 6.18

Suppose that there are two types of philosophers. One type always picks up his left fork first (a "lefty"), and the other type always picks up his right fork first (a "righty"). The behavior of a lefty is defined in Figure 6.12. The behavior of a righty is as follows:

```
begin
  repeat
       think;
       wait ( fork[ (i+1) mod 5] );
       wait ( fork[i] );
       eat;
       signal ( fork[i] );
       signal ( fork[ (i+1) mod 5] );
  forever
end;
```

Prove the following:

a) Any seating arrangement of lefties and righties with at least one of each avoids deadlock.
b) Any seating arrangement of lefties and righties with at least one of each prevents starvation.

Answer 6.18

a) Assume that every philosopher is waiting for the fork on their right. This is deadlock. Now if only one philosopher waits for the fork on the left. The fork on the right of this philosopher is therefore available. This would mean that the philosopher to their right would be free to eat. If we increase the amount of left hand eaters there will always be at least one fork available for one philosopher to finish eating. Once all the philosophers are left handed, we reach deadlock. However, this would contradict the existence of at least one right handed eater. Therefore, deadlock is not possible.

b) Assume that philosopher Pi1 is starving. This means that he is waiting to eat. Let's assume the Pi1 is a left-handed eater. Let's also assume that Pi1 is not holding any fork. This means that Pi2 (on Pi1's left) is holding his right fork. If all philosophers are starving then they all must be right-handed continuing this logic around the table. However, Pi1 is left-handed which is a contradiction.
Now let's assume all the same conditions except that Pi1 is holding one fork (his left fork). This means that Pi3 (on Pi1's right) is holding his left fork. If all philosophers are starving then they all must be left-handed continuing this logic around the table. However, Pi1 is right handed which is a contradiction.

Question 7.10

The Fibonacci sequence is defined as follows:

$$F_0 = 0, \quad F_1 = 1, \quad F_{n+2} = F_{n+1} + F_n, \quad n \geq 0$$

a) Could this sequence be used to establish a buddy system?
b) What would be the advantage of this system over the binary buddy system described in this chapter?

Answer 7.10

a) Yes, the block sizes could satisfy $F_n = F_{n-1} + F_{n-2}$.
b) This scheme offers more block sizes than a binary buddy system. This decreases the potential for internal fragmentation. However, this also increases potential for external fragmentation since many useless small blocks are created.

Question 7.12

Consider a simple paging system with the following parameters: 2^{32} bytes of physical memory; page size of 2^{10} bytes; 2^{16} pages of logical address space.

a) How many bits are in a logical address?
b) How many bytes in a frame?
c) How many bits in the physical address specify the frame?
d) How many entries in the page table?
e) How many bits in each page table entry? Assume each page table entry contains a valid/invalid bit.

Answer 7.12

a) The number of bytes in the logical address space is $2^{16} \times 2^{10} = 2^{26}$ bytes. So, 26 bits are required for the logical address.
b) A frame is the same size as a page (2^{10} bytes).
c) The number of frames in main memory is $2^{32} / 2^{10} = 2^{22}$ frames. So, 22 bits are needed to specify the frame.
d) There is one entry for each page in the logical address space. So, there are 2^{16} entries.
e) 22 bits are needed to specify the frame location in main memory excluding the valid / invalid bit. So, in total there are 23 bits.

Question 7.13

Write the binary translation of the logical address 0001010010111010 under the following hypothetical memory management schemes, and explain your answer:

a) a paging system with a 256-address page size, using a page table in which the frame number happens to be four times smaller than the page number
b) a segmentation system with a 1K-address maximum segment size, using a segment table in which bases happen to be regularly placed at real addresses: 22 + 4,096 + segment #

Answer 7.13

a) 00010100 is the page number.
10111010 is the page offset.

From the page lookup table 00010100 becomes 20 / 4 = 5.

The page frame number is 00000101.

The new logical address is 0000010110111010.

b) 10 bits is needed to represent the segment offset.
6 bits are left to represent the address.

0010111010 is the segment offset.
000101 is the address.

186 is the segment offset.
5 is the address.

The base address is 22 + 4096 + 5 = 4123.
The physical address is then 4123 + 186 = 4309.

1000011010101 is the physical address.

Question 7.14

Consider a simple segmentation system that has the following segment table:

Starting Address	Length (bytes)
660	248
1,752	422
222	198
996	604

For each of the following logical addresses, determine the physical address or indicate if a segment fault occurs:

a) 0, 198
b) 2, 156
c) 1, 530
d) 3, 444
e) 0, 222

Answer 7.14

a) 660 + 198 = 858
b) 222 + 156 = 378

c) Segmentation Fault.
d) 996 + 444 = 1440
e) 660 + 222 = 882

Question 8.1

Suppose the page table for the process currently executing on the processor looks like the following. All numbers are decimal, everything is numbered starting from zero, and all addresses are memory byte addresses. The page size is 1,024 bytes.

Virtual page number	Valid bit	Reference bit	Modify bit	Page frame number
0	1	1	0	4
1	1	1	1	7
2	0	0	0	–
3	1	0	0	2
4	0	0	0	–
5	1	0	1	0

a) Describe exactly how, in general, a virtual address generated by the CPU is translated into a physical main memory address.
b) What physical address, if any, would each of the following virtual addresses correspond to? (Do not try to handle any page faults, if any.
   i) 1,052
   ii) 2,221
   iii) 5,499

Answer 8.1

a) Split the binary number into a virtual page address and an offset. Extract the page frame number, use the VPN as an index to the page frame table. Concatenate the page frame with the offset to get the physical address.
b) See below.
   i) 1052 % 1024 = 28
      1052 / 1024 = 1, VPN = 1, PFN = 7
      PA = 7 * 1024 + 28 = 7196
   ii) 2221 % 1024 = 173
      2221 / 1024 = 2, VPN = 2, PFN = _
      PA = Segmentation Fault.
   iii) 5499 % 1024 = 379
      5499 / 1024 = 5, VPN = 5, PFN = 0
      PA = 0 * 1024 + 379 = 379

Question 8.2

Consider the following program.

3SH3 HOMEWORK 3
Bradley Kohler

```
#define Size 64
int A[Size; Size], B[Size; Size], C[Size; Size];
int register i, j;
for (j = 0; j< Size; j ++)
  for (i = 0; i< Size; i++)
C[i; j] = A[i; j] + B[i; j];
```

Assume that the program is running on a system using demand paging and the page size is 1 Kilobyte. Each integer is 4 bytes long. It is clear that each array requires a 16-page space. As an example, A[0, 0]-A[0, 63], A[1, 0]-A[1, 63], A[2, 0]A[2, 63], and A[3, 0]-A[3, 63] will be stored in the first data page. A similar storage pattern can be derived for the rest of array A and for arrays B and C. Assume that the system allocates a 4-page working set for this process. One of the pages will be used by the program and three pages can be used for the data. Also, two index registers are assigned for i and j (so, no memory accesses are needed for references to these two variables).

a) Discuss how frequently the page fault would occur (in terms of number of times C[i, j] = A[i, j] + B[i, j] are executed).
b) Can you modify the program to minimize the page fault frequency?
c) What will be the frequency of page faults after your modification?

Answer 8.2

a) Each array requires 16 pages since 64 * 64 * 4 = 16384 and 16384 % 1024 = 16. The working set for program data is 3 pages, meaning that 3 pages can be in main memory at a time. The working set will correspond to page 1 of A, page 1 of B and page 1 of C. However, A only has A[0,0] – A[3,63] in the first page. So, the second page of A is not in main memory and we will encounter a page fault every four iterations of the inner loop.
b) If the order of nested loops is changed then page faults will occur less frequently.
c) All arithmetic operations from C[0,0] = A[0,0] – B[0,0] to C[3,63] = A[3,63] - B[3,63] will execute before a page fault. A page fault will occur ever 256 iterations of the inner loop.

Question 8.4

Consider the following string of page references 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2.

Complete a figure similar to Figure 8.14, showing the frame allocation for:

a) FIFO (first-in-first-out)
b) LRU (least recently used)
c) Clock
d) Optimal (assume the page reference string continues with 1, 2, 0, 1, 7, 0, 1)
e) List the total number of page faults and the miss rate for each policy. Count page faults only after all frames have been initialized.

Answer 8.4

a) FIFO

3SH3 HOMEWORK 3
Bradley Kohler

Stream	7	0	1	2	0	3	0	4	2	3	0	3	2
	7	7	7	2	2	2	2	4	4	4	0	0	0
		0	0	0	0	3	3	3	2	2	2	2	2
			1	1	1	1	0	0	0	3	3	3	3
				F		F	F	F	F	F	F		

b) LRU

Stream	7	0	1	2	0	3	0	4	2	3	0	3	2
	7	7	7	2	2	2	2	4	4	4	0	0	0
		0	0	0	0	0	0	0	0	3	3	3	3
			1	1	1	3	3	3	2	2	2	2	2
				F		F		F	F	F	F		

c) CLOCK

Stream	7	0	1	2	0	3	0	4	2	3	0	3	2
	7*	7*	7*	2*	2*	2*	2*	4*	4*	4*	0*	0	0
		0*	0*	0	0	0*	0*	0	2*	2*	2	2	2
			1*	1	1	3*	3*	3	3	3*	3	3	3
				F		F		F	F		F		

d) OPT

Stream	7	0	1	2	0	3	0	4	2	3	0	3	2
	7	7	7	2	2	2	2	2	2	2	2	2	2
		0	0	0	0	0	0	4	4	4	0	0	0
			1	1	1	3	3	3	3	3	3	3	3
				F		F		F			F		

Question 8.11

Consider a system with memory mapping done on a page basis and using a singlelevel page table. Assume that the necessary page table is always in memory.

a) If a memory reference takes 200 ns, how long does a paged memory reference take?
b) Now we add an MMU that imposes an overhead of 20 ns on a hit or a miss. If we assume that 85% of all memory references hit in the MMU TLB, what is the Effective Memory Access Time (EMAT)?
c) Explain how the TLB hit rate affects the EMAT.

Answer 8.11

a) 400 nanoseconds. 200 to get the entry from the page table. 200 to access the mem location.
b) (200 + 20) * 0.85 + (400 + 20) * 0.15 = 250ns
c) The higher the TLB hit rate is the smaller the EMAT. There is an additional 200ns penalty for a miss.

3SH3 HOMEWORK 3
Bradley Kohler

Question 8.15

Consider the following sequence of page references (each element in the sequence represents a page number):

1 2 3 4 5 2 1 3 3 2 3 4 5 4 5 1 1 3 2 5

…

a) Draw a diagram similar to that of Figure 8.17 for the reference sequence just defined for the values Δ = 1, 2, 3, 4, 5, 6.
b) Plot s20(Δ) as a function of Δ.
c) Plot m20(Δ) as a function of Δ.

Answer 8.15

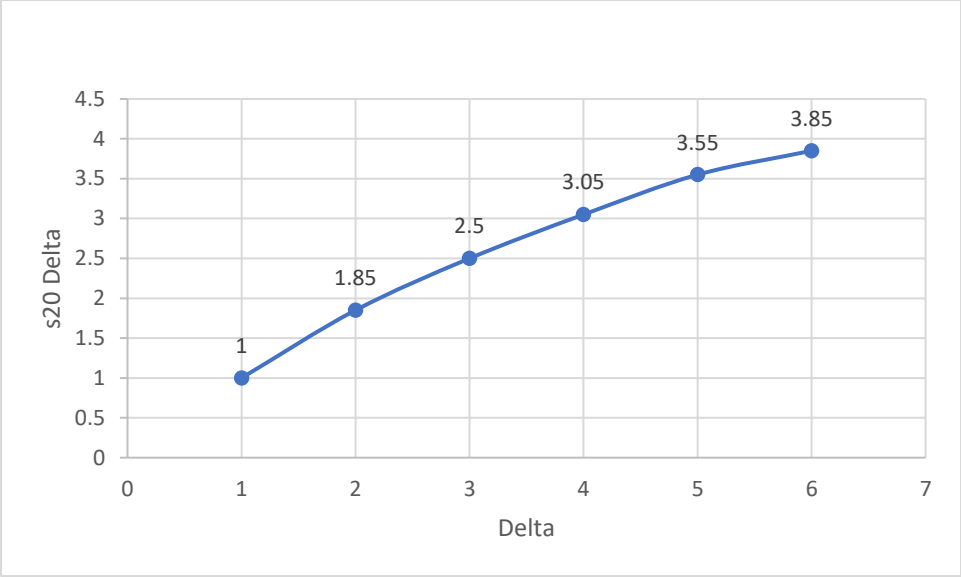
a) See below.

	Window Size					
Seq	1	2	3	4	5	6
1	1	1	1	1	1	1
2	2	1 2	1 2	1 2	1 2	1 2
3	3	2 3	1 2 3	1 2 3	1 2 3	1 2 3
4	4	3 4	2 3 4	1 2 3 4	1 2 3 4	1 2 3 4
5	5	4 5	3 4 5	2 3 4 5	1 2 3 4 5	1 2 3 4 5
2	2	5 2	4 5 2	3 4 5 2	3 4 5 2	1 3 4 5 2
1	1	2 1	5 2 1	4 5 2 1	3 4 5 2 1	3 4 5 2 1
3	3	1 3	2 1 3	5 2 1 3	4 5 2 1 3	4 5 2 1 3
3	3	3	1 3	2 1 3	5 2 1 3	4 5 2 1 3
2	2	3 2	3 2	1 3 2	1 3 2	5 1 3 2
3	3	2 3	2 3	2 3	1 2 3	1 2 3
4	4	3 4	2 3 4	2 3 4	2 3 4	1 2 3 4
5	5	4 5	3 4 5	2 3 4 5	2 3 4 5	2 3 4 5
4	4	5 4	5 4	3 5 4	2 3 5 4	2 3 5 4
5	5	4 5	4 5	4 5	3 4 5	2 3 4 5
1	1	5 1	4 5 1	4 5 1	4 5 1	3 4 5 1
1	1	1	5 1	4 5 1	4 5 1	4 5 1
3	3	1 3	1 3	5 1 3	4 5 1 3	4 5 1 3
2	2	3 2	1 3 2	1 3 2	5 1 3 2	4 5 1 3 2
5	5	2 5	3 2 5	1 3 2 5	1 3 2 5	1 3 2 5

b) $S_{20}(1) = 1$
…
See below.

3SH3 HOMEWORK 3
Bradley Kohler

Δ	1	2	3	4	5	6
$s_{20}(\Delta)$	1	1.85	2.5	3.05	3.55	3.85



c) S_{20} (1) = 0.9

…

See below.

Δ	1	2	3	4	5	6
$s_{20}(\Delta)$	0.90	0.85	0.75	0.65	0.55	0.50