Operating Systems Module: CO-562
Constructor University Date: 2026-10-23
Dr. Jürgen Schönwälder Due: 2026-10-30

Problem Sheet #8

Problem 8.1: *positioning algorithms* (1+1+1+1 = 4 points)

A system using segmentation has free blocks of the following sizes:

```
17 KiB,  8 KiB, 10 KiB, 21 KiB, 12 KiB, 13 KiB
```

The blocks are ordered as shown above and the last memory allocation took place right before the first block. The system receives the following sequence of memory requests (in the order shown below):

```
11 KiB,  9 KiB,  7 KiB, 16 KiB
```

If a free block is allocated that is bigger than the request, then a new small block is left over and positioned at the same place where the bigger block was positioned in the list of free blocks.

a) Show how the list of free blocks changes after each allocation using the best-fit algorithm.

```
         17 KiB,  8 KiB, 10 KiB, 21 KiB, 12 KiB, 13 KiB
11 KiB:
 9 KiB:
 7 KiB:
16 KiB:
```

b) Show how the list of free blocks changes after each allocation using the worst-fit algorithm.

```
         17 KiB,  8 KiB, 10 KiB, 21 KiB, 12 KiB, 13 KiB
11 KiB:
 9 KiB:
 7 KiB:
16 KiB:
```

c) Show how the list of free blocks changes after each allocation using the first-fit algorithm.

```
         17 KiB,  8 KiB, 10 KiB, 21 KiB, 12 KiB, 13 KiB
11 KiB:
 9 KiB:
 7 KiB:
16 KiB:
```

d) Show how the list of free blocks changes after each allocation using the next-fit algorithm.

```
         17 KiB,  8 KiB, 10 KiB, 21 KiB, 12 KiB, 13 KiB
11 KiB:
 9 KiB:
 7 KiB:
16 KiB:
```

Problem 8.2: *page replacement algorithms* (1+1+1 = 3 points)

An operating system uses demand paging to implement virtual memory. A user-space process uses four pages, numbered 1 to 4. The pages are accessed in the following order (reference string): 1, 4, 2, 3, 4, 4, 1, 3, 2, 1. None of the pages used by the user-space process are resident when the process starts and the frames allocated to the process are unused. Fill out the following tables by writing page numbers into the cells. Each number indicates which page a frame holds after accessing the page indicated by the reference string. Write down the number of page faults for each scenario.

a) Show how the pages are mapped to two frames using the First-In-First-Out (FIFO) page replacement algorithm.

reference string	1	4	2	3	4	4	1	3	2	1
frame 0										
frame 1										

Show how the pages are mapped to three frames using the First-In-First-Out (FIFO) page replacement algorithm.

reference string	1	4	2	3	4	4	1	3	2	1
frame 0										
frame 1										
frame 2										

b) Show how the pages are mapped to two frames using Belady's Optimal (BO) page replacement algorithm.

reference string	1	4	2	3	4	4	1	3	2	1
frame 0										
frame 1										

Show how the pages are mapped to three frames using Belady's Optimal (BO) page replacement algorithm.

reference string	1	4	2	3	4	4	1	3	2	1
frame 0										
frame 1										
frame 2										

c) Show how the pages are mapped to two frames using the Least Recently Used (LRU) page replacement algorithm.

reference string	1	4	2	3	4	4	1	3	2	1
frame 0										
frame 1										

Show how the pages are mapped to three frames using the Least Recently Used (LRU) page replacement algorithm.

reference string	1	4	2	3	4	4	1	3	2	1
frame 0										
frame 1										
frame 2										

Hint: Do not move pages between frames except when necessary.

Problem 8.3: *paging and page tables* (1+1+1 = 3 points)

Consider a tiny computer system with a physical memory space of 16 frames. Each 12-bit logical address uses 4 bits for the page number and 8 bits for the offset within a page. There are two processes P_1 and P_2 with the logical address spaces shown below.

Process P_1		
Page	Logical Addresses	Segment
$p_{1,0}$	0x000-0x0FF	text
$p_{1,1}$	0x100-0x1FF	text
$p_{1,2}$	0x200-0x2FF	data
$p_{1,5}$	0x500-0x5FF	heap
$p_{1,6}$	0x600-0x6FF	stack
$p_{1,8}$	0x800-0x8FF	stack

Process P_2		
Page	Logical Addresses	Segment
$p_{2,0}$	0x000-0x0FF	text
$p_{2,1}$	0x100-0x1FF	text
$p_{2,4}$	0x400-0x4FF	data
$p_{2,5}$	0x500-0x5FF	data
$p_{2,6}$	0x600-0x6FF	heap
$p_{2,8}$	0x800-0x8FF	stack

Some pages reside in physical memory as shown in the table below. The notation $p_{i,n}$ refers to page n of the logical address space of process P_i. The OS pages are used by the operating system, unused frames are marked with a dash.

Frame	Physical Addresses	Loaded Page
0	0x000-0x0FF	OS
1	0x100-0x1FF	$p_{2,5}$
2	0x200-0x2FF	-
3	0x300-0x3FF	$p_{1,8}$
4	0x400-0x4FF	$p_{2,4}$
5	0x500-0x5FF	-
6	0x600-0x6FF	$p_{1,1}$
7	0x700-0x7FF	-
8	0x800-0x8FF	$p_{1,0}$
9	0x900-0x9FF	$p_{2,0}$
10	0xA00-0xAFF	-
11	0xB00-0xBFF	$p_{1,6}$
12	0xC00-0xCFF	$p_{2,1}$
13	0xD00-0xDFF	-
14	0xE00-0xEFF	-
15	0xF00-0xFFF	-

a) Write down the page tables for both processes P_1 and P_2. Each page table entry maintains the following additional bits: r = read access, w = write access, x = execute access, d = dirty, v = valid, commonly written in the form `rwxdv` if all bits are set or as `rw--v` if only the r, w, and v bits are set. Assume that all writable pages are dirty.

b) The CPU executes process P_1 and the machine instructions modify a global variable and a dynamically allocated string. A context switch occurs and the CPU executes process P_2, which performs a function call that allocates and initializes 16 bytes on the heap.

Write down the content of the page tables after all write operations and initializations have been performed. If a page fault occurs use the first free physical frame to load the page.

c) The processes P_1 and P_2 establish a shared memory page to exchange data. The shared page appears as $p_{1,4}$ in the logical address space of P_1 and as $p_{2,2}$ in the logical address space of P_2. Show the resulting use of the physical memory frames (i.e., update the table shown above).