Name _____

CS 273 OPTIONAL Second Exam R. Brown November 9, 2020
SHOW YOUR WORK—No work may mean no credit

I pledge my honor that I have neither given nor received assistance during this exam, and that I have seen no dishonest work.

Signed _____

☐ I have intentionally not signed the pledge (*check only if appropriate*)

(*6 pts*) 1. A computer has four page frames. The time of loading, time of last access and the R and M bits for each page are as shown below (times are in clock ticks):

Page	Loaded	Last ref.	R	M
0	114	137	1	1
1	102	120	0	1
2	113	130	0	0
3	128	138	1	0

a) Which page will NRU replace, and why?

b) Which page will LRU replace, and why?

c) Which page will FIFO/second chance replace, and why?

d) For **one** of the following terms, (i) describe the meaning of that operating-system term, and (ii) give an example of that concept and/or a figure illustrating that concept.

page table *or* internal fragmentation

(*6 pts*) 2. The six layers of I/O software are:

(A) User-level software
(B) Device-independent OS software
(C) Device driver
(D) Interrupt handler
(E) Device controller
(F) Device

Which of the six layers of I/O software **must** be used in the following operations?

a) A new file is saved to disk for the first time by `emacs`

b) A page is swapped out due to a page fault in kernel code

c) A program's file read operation is satisfied by a disk block that is already present in memory (i.e., in the block cache)

d) For question a) above, explain why your answer does/doesn't include **two** of the following:

(C) Device driver

(D) Interrupt handler

(E) Device controller

(*6 pts*) 3. Suppose that the following actions happen in the indicated order:

i. Process A acquires exclusive access to a printer.
ii. Process B acquires exclusive access to a tape drive, then blocks trying to get access to that printer.
iii. Process A then blocks trying to get access to that tape drive.

a) Draw a resource graph indicating the relationships between processes and resources in this situation. (Use circles for processes, squares for resources, and arrows to indicate relationships between these.)

b) Does this situation constitute a deadlock? Justify your answer using **either** the four deadlock conditions (mutual exclusion, hold and wait, no preemption, circular wait) **or** the definition of deadlock.

c) **If there is a deadlock in b)**, explain in the space below how that deadlock could have been **avoided**.
If there is no deadlock in b), modify your diagram in part a) to **add** processes, resources, and/or **dashed (or other-colored)** arrows that would create a deadlock.

(*6 pts*) 4. Consider the example socket program `receiver.c` shown on the next page.

a) What line number(s) cause a message to be received from a "sender" program?
Also, draw a box labelled **a** on the next page for these lines.

b) What line number(s) create a new socket?
Also, draw a box labelled **b** on the next page for these lines.

c) What line number(s) set up a new server socket? (Include any code that sets up relevant data structures, makes relevant system calls, etc.
Also, draw a box labelled **c** on the next page for these lines.

d) What line number(s) make a system call that assigns a port to a socket?
Also, draw a box labelled **d** on the next page for these lines.

e) How would you modify this code to send a 3-byte message `ACK` back to the "client" program?
- You may write the new code below **or** on the code page at the end.
- **Draw an arrow in the code on the next page page** to indicate where the new code should be inserted. Draw multiple arrows if more than one insertion is needed, and indicate what should be inserted where.
- Here is a spec for the system call `send`, in case you want to use it in your answer. (**Use 0 for the fourth argument** of `send()` for this problem.)

`send`
4 Arguments: A `int` representing a socket descriptor, an array of `char` (type `char*`), an integer (type `size_t`) indicating the number of bytes of *arg2* to send, and an `int` indicating option flags.
State change: The first *arg3* bytes of the array *arg2* are sent on socket descriptor *arg1* with option flags *arg4*.
Return: An `int` value which is either the non-negative number of bytes sent (on success), or a negative number indicating an error.

```
 1: /* header files omitted */
 2: #define MAXBUFF 100
 3:
 4: int main(int argc, char **argv) {
 5:   char *prog = argv[0];
 6:   int port;
 7:   int serverd;  /* socket descriptor for receiving new connections */
 8:   port = atoi(argv[1]);   /* retrieve port number from first command-line arg */
 9:
10:   if ((serverd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
11:     printf("%s ", prog);
12:     perror("socket()");
13:     return 1;
14:   }
15:   struct sockaddr_in sa;
16:   sa.sin_family = AF_INET;
17:   sa.sin_port = htons(port);
18:   sa.sin_addr.s_addr = INADDR_ANY;
19:   if (bind(serverd, (struct sockaddr*) &sa, sizeof(sa)) < 0) {
20:     printf("%s ", prog);
21:     perror("bind()");
22:     return 1;
23:   }
24:   if (listen(serverd, 5) < 0) {
25:     printf("%s ", prog);
26:     perror("listen()");
27:     return 1;
28:   }
29:
30:   int clientd;  /* socket descriptor for communicating with client */
31:   struct sockaddr_in ca;
32:   int size = sizeof(struct sockaddr);
33:   printf("Waiting for a incoming connection...\n");
34:   if ((clientd = accept(serverd, (struct sockaddr*) &ca, &size)) < 0) {
35:     printf("%s ", prog);
36:     perror("accept()");
37:     return 1;
38:   }
39:   char buff[MAXBUFF];  /* message buffer */
40:   int ret;  /* return value from a call */
41:   if ((ret = recv(clientd, buff, MAXBUFF-1, 0)) < 0) {
42:     printf("%s ", prog);
43:     perror("recv()");
44:     return 1;
45:   }
46:   buff[ret] = '\0';  // add terminating nullbyte to received array of char
47:   printf("Received message (%d chars):\n%s\n", ret, buff);
48:   if ((ret = close(clientd)) < 0) {
49:     printf("%s ", prog);
50:     perror("close(clientd)");
51:     return 1;
52:   }
53:   if ((ret = close(serverd)) < 0) {
54:     printf("%s ", prog);
55:     perror("close(serverd)");
56:     return 1;
57:   }
58:   return 0;
59: }
```