Spring Semester 2024

KAIST EE209

Programming Structures for Electrical Engineering

Final Exam

Date: 2024.06.10

Time: 13:00 ~ 15:30

Student ID:

Name:

The exam is closed book and notes. Read the questions carefully and focus your answers on what has been asked. You are allowed to ask the instructor/TAs for help only in understanding the questions, in case you find them not completely clear. Be concise and precise in your answers and state clearly any assumption you may have made. All your answers must be included in the attached sheets. You have 150 minutes to complete your exam. Be wise in managing your time.
Please do not fill in the "Score" fields below. Self-grading is not allowed. Good luck!

1	/15
2	/10
3	/10
4	/10
5	/15
6	/20
7	/10
Total	/100

Student ID: Name:

1. (15 pt) Hash table

   1.1 (5 pt) The code below describes the 'hash table' structure we learned in the lecture.

```
enum {BUCKET_COUNT = 1024};

struct Node {
   const char *key;
   int value;
   struct Node *next;
};

struct Table {
   struct Node *array[BUCKET_COUNT];
};

void Table_add(struct Table *t, const char *key, int value)
{
   struct Node *p = (struct Node*)malloc(sizeof(struct Node));
   int h = hash(key);
   p->key =(const char*)malloc(strlen(key) + 1);
   strcpy(p->key, key);
   p->value = value;
   p->next = t->array[h];
   t->array[h] = p;
}
```

Now, implement `Table_delete` function when the data structure owns a copy of the key. Hint: As we've learned in modularity, well-designed module manages resource consistently (i.e., a module should free a resource if and only if the module has allocated that resource). The function should also satisfy the below requirements:

• Return 1 if the node is successfully removed.
• Return 0 if the node with the corresponding key is not found.
• After deletion, all other nodes should remain accessible and should be in the same order.
• Assume there are **no duplicate keys**.
• You may use library functions (e.g., `int strcmp(const char *string1, const char *string2)` which returns 0 when string1 is equal to string2).
• You can use `hash` function. But you may NOT call other hash table functions (e.g., `Table_search`).

```
int Table_delete(struct Table *t1, const char *key)
{
   struct Node *p, *p_prev = NULL;
   int h = hash(key);
   for (p = t1->array[h]; p != NULL; p = p->next) {
   Ans)
      if (strcmp(key, p->key) == 0) {
         if (p_prev == NULL) {
            t1->array[h] = p->next;
         } else {
            p_prev->next = p->next;
         }
         free(p->key);
         free(p);
         return 1;
      }
      p_prev = p;




   }
   return 0;
}
```

grading criteria

- any syntax error or minor error: -0.5pts
- logic error
    - does not consider when p_prev is NULL or does not consider when p_prev is not NULL (-1pt)
    - others (-1pt)
- no free(p->key): -0.5pts
- no free (p): -0.5pts
- incomplete code but writes some related functions: only 1 pt

1.2 (5 pt) Implement `Table_subtract` function, which deletes the nodes contained in table ‘`t1`’ if their keys are also present in table ‘`t2`’. The function should return the number of nodes deleted from t1. You have to use the `Table_delete` function implemented earlier in this question.

```
int Table_subtract(struct Table *t1, struct Table *t2)
{
   struct Node *p;
   int b, count = 0;
   for (b = 0; b < BUCKET_COUNT; b++) {
   Ans)
      for (p = t2->array[b]; p != NULL; p = p->next;) {
         count += Table_delete(t1, p->key);
      }



   }
   return count;
}
```

grading criteria

- any syntax error or minor error: -0.5pt
- logic error
    - did not use Table_delete() (-1pt)
    - did not rotate the list of each bucket. (-1pt)
    - others (-1pt)
- incomplete code but writes some related functions: only 1 pt

1.3 (5 pt) Implement `Table_merge` function, which adds the nodes contained in table ‘`t2`’ to ‘`t1`’ if their keys are not already present in table ‘t1’. Also, the function should return the number of nodes added to t1. You have to use `Table_add` and `Table_search` function in this question. (Hint: `Table_search` function takes three parameters, (`struct Table *t, const char *key, int *value`) and returns 1 if the input ‘`key`’ exists in hash table `t`, otherwise, it returns 0. If the ‘`key`’ exists in the table ‘`t`’, the value of the corresponding node is copied to the input integer pointer ‘`value`’.)

```
int Table_merge(struct Table *t1, struct Table *t2)
{
   struct Node *p;
   int b, value, count = 0;
   for (b = 0; b < BUCKET_COUNT; b++) {
   Ans)
```

```
      for (p = t2->array[b]; p != NULL; p = p->next;) {
         if (Table_search(t1, p->key, &value) == 0) {
            Table_add(t1, p->key, p->value);
            count++;
         }
      }


   }
   return count;
}
```

grading criteria

- any syntax error or minor error: -0.5pts
- logic error
    - did not use table_search or table_add (-1pt)
    - did not rotate the list of each bucket (-1pt)
- incomplete code but writes some related functions: only 1 pt

Student ID: Name:

2. (10 pt) Assembly

2.1 (5 pt) Given the following assembly code, re-construct the C code that produced it.

```
.section .rodata
.LC1:
    .string "%lu\n"

mystery1:
0x00400566 <+0>:     cmp      $0, %rsi
0x00400569 <+3>:     jle      0x400599 <mystery1+51>
0x0040056b <+5>:     push     %rbp
0x0040056c <+6>:     push     %rbx
0x0040056d <+7>:     sub      $0x8, %rsp
0x00400571 <+11>:    mov      %rsi, %rbx
0x00400574 <+14>:    mov      %rdi, %rbp
0x00400577 <+17>:    sarl     $1, %rbx
0x0040057a <+20>:    callq    0x400566 <mystery1>
0x0040057f <+25>:    mov      -0x8(%rbp, %rbx, 8), %rsi
0x00400584 <+30>:    mov      $.LC1, %edi
0x00400589 <+35>:    mov      %0x0, %eax
0x0040058e <+40>:    callq    0x400430 <printf>
0x00400593 <+45>:    add      $0x8, %rsp
0x00400597 <+49>:    pop      %rbx
0x00400598 <+50>:    pop      %rbp
0x00400599 <+51>:    ret
```

```
void mystery1(long *arr, size_t count) {
    if (____________________) {                  // line 1
        ____________________;                    // line 2
        printf("%lu\n", arr[count - 1] );     // line 3
    }
}
```

Answer:

```
void mystery1(long *arr, size_t count) {
    if (      count > 0       ) {                // line 1
        mystery1(arr, count / 2);                // line 2
        printf("%lu\n",  arr[count - 1]);        // line 3
    }
}
```

grading criteria

wrong answer → -2 (all wrong answer → get 0 in total)

wrong answer in right form, ex) mystery1(arr, count) → -1

2.2 (5 pt) Given the following assembly code, re-construct the C code that produced it.

```
int mystery2(char *param1, int *param2, int param3) {
    int local = strlen(param1);                    // line 1
    for (int i = rand(); i < 0 ; i += 4) {         // line 2
        *param2 -= i;                              // line 3
        local = 5*local + 14;                      // line 4
     }
    return -mystery2(NULL,&local,param2[1]*param3); // line 5
}
```

```
mystery2:
    push     %rbp
    push     %rbx
    sub      $0x18,%rsp
    mov      %rsi,%rbx
    mov      %edx,%ebp
    callq    <strlen>
    mov      %eax,0xc(%rsp)
    callq    <rand>
    jmp      .L2
.L1:
    sub      %eax,____(a)____
    mov      0xc(%rsp),%ecx
    lea      0xe(%rcx,%rcx,4),%ecx
    mov      %ecx,0xc(%rsp)
    ________(b)________
.L2:
    ________(c)________
    jl       .L1
    mov      %ebp,%edx
    imul     0x4(%rbx),%edx
    lea      0xc(%rsp),%rsi
    mov      0x0,%edi
    callq    <mystery2>
    neg      eax
    add      0x18,%rsp
    pop      rbx
    pop      rbp
    ret
```

Answer

(a) : **`(%rbx)`** , (b): **`add $0x4,%eax`** , (c) : **`cmp $0,%eax`**

```
mystery2:
    push    %rbp
    push    %rbx
    sub     $0x18,%rsp
    mov     %rsi,%rbx
    mov     %edx,%ebp
    callq   <strlen>
    mov     %eax,0xc(%rsp)
    callq   <rand>
    jmp     .L2
.L1:
    sub     %eax,(%rbx)
    mov     0xc(%rsp),%ecx
    lea     0xe(%rcx,%rcx,4),%ecx
    mov     %ecx,0xc(%rsp)
    add     $0x4,%eax
.L2:
    cmp     $0,%eax
    jl      .L1
    mov     %ebp,%edx
    imul    0x4(%rbx),%edx
    lea     0xc(%rsp),%rsi
    mov     0x0,%edi
    callq   <mystery2>
    neg     eax
    add     0x18,%rsp
    pop     rbx
    pop     rbp
    ret
```

grading criteria

Only give the points with exact answers

(a) 2

(b) 2

(c) 1

Student ID: Name:

3. (10 pt) Exception

3.1 (2 pt) There are two types of exception: trap and interrupt. Which exception type corresponds to (a) and (b)?
(a) occurs when a user presses a key on the keyboard.
(b) occurs when program requests heap memory.

Answer: (a):interrupt, (b): trap
Guide for grading - 1 points for each exception type

3.2 (2 pt) Following code denotes the situation where the exception handler kills the application when the exception occurs. Among the following, select the exception(s) corresponding to the case of the following code.

```
int *ptr;
ptr = 0;
*ptr = 1;
```

1. segmentation fault
2. IO completion
3. system call
4. divide-by-zero

Answer: 1

3.3 (6 pt) When the user program calls a system call, the trap instruction is executed and the execution mode changes from the user mode to the kernel mode. It is called "mode switch". When the OS switches the CPU from one program to another, the OS saves the registers of the old process and loads the register values for the new program. Please specify whether the following situation corresponds to mode switch, context switch or both.

a) privilege level changes (2pt): mode switch.
b) current address map (page table) changes (2pt):context switch
c) current register values are saved (2pt):both

Explanation

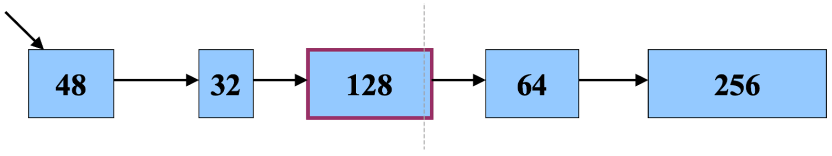
a) The privilege level changes when changing from user mode to kernel mode. During context switch, one process in kernel model is changed to another process in kernel mode, so the privilege level does not change.

b) A process has a single page table. The page table of a process contains address maps of both user mode and kernel mode. So the page table does not changes during mode switch.
   In case of context switch, the process is changed so the page table is also changed.
c) During mode switch, a process in user mode saves its register values to the kernel stack and turns into the kernel mode.
   During context switch, one process in kernel mode saves its register values in the memory and is switched to other process.

Student ID: Name:

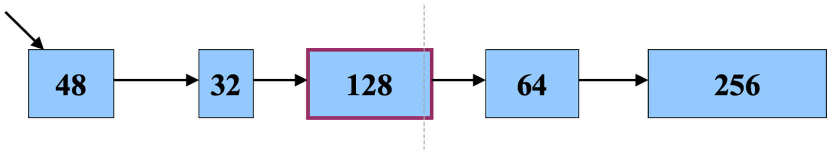
4. (10 pt) Memory Allocation

4.1 (3 pt) Consider the list of free memory chunks below. Show the free memory chunks after "allocate (64)" using "first-fit".



48->32->64->64->256

4.2 (3 pt) Consider the list of free memory chunks below. Show the free memory list after "allocate (64)" using "best-fit".



48->32->128->256

4.3 (4 pt) Consider the following code. If you run this code, what is going to happen? Is it going to run forever? Or is the operating system going to kill the program at some point? To get the full credit, provide detailed reasoning.

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

int main(){
   while (1) {
      malloc (0) ;
   }
   return 0;
}
```

answer:
malloc (0) allocates a minimal allocation unit of memory. Thus, the memory is exhausted at the end. This results in the corresponding process to be killed or crashed. .

grading criteria
- correct answer without detailed reasoning. 2pt

Student ID: Name:

5. (15 pt) IO
Consider the following text file “txt”. It contains 5 lines of string written in text editor (vi, emacs, nano). EOF means end of file.

```
012345
123456
234567
345678
456789
EOF
```

5.1 (5 pt) What is the output of this program? Please explain the reason. Assume that there is no runtime error.

```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>

int main(int argc, char** argv) {
  char buf[10];
  int fd, i;
  fd = open("txt", O_RDONLY, 0640);

  for (i = 0; i < 5; i++) {
    read(fd, buf, 4);
    printf("%c", buf[0]);
  }
  return 0;
}
```

Answer: 04264
04151 is also correct if ONLY the student wrote that “new line can be 2 bytes, **carriage return (CR)** and **line feed (LF),** or **“\r\n”**”

Explanation
New line character (‘\n’) takes 1 byte.

Guide for grading - no partial point; no deduction even if no explanation (except 04151)

5.2 (5 pt) What is the output of the following program?

```
#include <stdio.h>

int main(int argc, char** argv) {
  char buf1[10], buf2[10];
  int n1, n2;
  FILE *fp = fopen("txt", "r");

  fgets(buf1, 10, fp);
  fscanf(fp, "%d", &n1);
  fgets(buf2, 10, fp);
  fclose(fp);
  fp = fopen("txt", "r");
  fscanf(fp, "%d", &n2);
  printf("%s%d%s%d\n", buf1, n1, buf2, n2);

  return 0;
}
```

Answer:
012345
123456
12345

The answer should be written in 3 lines
“012345\n123456\n12345\n” is also correct (it needs 3 new line characters)

Explanation
There are 4 read operations.
1st read: buf1 = “012345\n”. buf1 contains one line.
2nd read: n1 = 123456. File pointer is now the end of the 2nd line (‘\n’).
3rd read: buf2 = “\n”. fgets() reads the remaining part of the 2nd line.
4th read: n2 = 12345. fp is closed and reopened, so n2 stores 012345 in “integer”.

Guide for grading - give 1 point for followings:
- 1st line starts with 012345
- 2nd line starts with 123456
- the last line ends with 12345 (or 12345\n)

5.3 (5 pt) Following is `man` page of `lseek()`.

NAME
lseek - reposition read/write file offset

SYNOPSIS

`#include <unistd.h>`
`off_t lseek(int` ***`fd`***, **`off_t`** ***`offset`***, **`int`** ***`whence`*`);`**

DESCRIPTION
lseek() repositions the file offset of the open file description associated with the file descriptor *fd* to the argument *offset* according to the directive *whence* as follows:

SEEK_SET
The file offset is set to *offset* bytes.

SEEK_CUR
The file offset is set to its current location plus *offset* bytes.

SEEK_END
The file offset is set to the size of the file plus *offset* bytes.

lseek() allows the file offset to be set beyond the end of the file. If data is later written at this point, subsequent reads of the data in the gap (a "hole") return null bytes ('\0') until data is actually written into the gap.

The following program creates a file “txt2”. Then, it writes some characters to “txt2”. When the program writes the characters to “txt2” file, it sets the current offset using `lseek()`. **What is the content of “txt2” after executing the program?** Assume that “txt2” did not exist at the beginning.

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>

int main(int argc, char** argv) {
  char buf[1024];
  FILE *fp = fopen("txt", "r");
  int fd2 = creat("txt2", 0640);  // create txt2

  for (int i = 1; i <= 2; i++) {
    fgets(buf, 1024, fp);         // read content of txt
    lseek(fd2, i, SEEK_END);
    write(fd2, buf, i);
  }
  lseek(fd2, 0, SEEK_SET);

  for (int i = 1; i <= 2; i++) {
    fgets(buf, 1024, fp);
    write(fd2, buf, i);
    lseek(fd2, i, SEEK_CUR);
  }

  return 0;
}
```

Answer: 203412

Explanation
After the first loop, txt2 becomes “H **0** H H **1 2**” (H means “hole”).
After the second loop, txt2 becomes “**2 0 3 4 1 2**”.

Guide for grading - give 1 point for followings:
- the answer has 6 digits
- 2nd, 5th, 6th digits are 0, 1 ,2, respectively.

Student ID: Name:

6. (20 pt) Process

6.1 (5 pt) What is the output of the following program? If there are more than one possible output, write all of them.

```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>

int main(int argc, char** argv) {
  int num = 0;
  pid_t pid;

  for (int i = 0; i < 3; i++) {
    if ((pid = fork()) == 0) {
      num++;
    }
    else {
      waitpid(pid, NULL, 0);
      printf("%d", num);
      return 0;
    }
  }
  return 0;
}
```

Answer: 210

Explanation
Let a process P1 forks to make child P2, P2 makes P3, and P3 makes P4. The value of “num” in P1, P2, P3, P4 is 0, 1, 2, 3, respectively.
1) P4 does not print anything and just return 0
2) P3 waits P4 and prints 2
3) P2 waits P3 and prints 1
4) P1 waits P2 and prints 0
Grandchildren are not children.

Guide for grading - no partial point; If the answer contains multiple outputs, no point

6.2 (5 pt) The following ‘p.c’ implements redirection. Fill in the blanks so that a command “./p [somepgm] [file1] [file2]” runs “[somepgm] < [file1] > [file2]”. You can write the answer in multiple lines.

```
// p.c

#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/wait.h>

int main(int argc, char** argv) {
  pid_t pid;
  if (argc < 4) return 0;

  pid = fork();
  if (pid == 0) {
    /* in child */
    int fd1, fd2;
    fd1 = open(_____A______, O_RDONLY, 0640);
    fd2 = creat(_____B______, 0640);

    ┌─────────────┐
    │      C      │
    └─────────────┘

    close(fd1);
    close(fd2);

    char *argv_new[] = {argv[1], NULL};
    execvp(argv[1], argv_new);
    fprintf(stderr, "exec failed\n");
    return -1;
  }
  /* in parent */
  pid = wait(NULL);
  return 0;
}
```

A:
B:
C:
Answer
A: argv[2]   B: argv[3]   C: close(0); close(1); dup(fd1); dup(fd2);

For A and B, “[file1]” and “[file2]” is also correct (both [] and “ are needed).
For C, there are multiple answers like followings:
  fclose(stdin); fclose(stdout); dup(fd1); dup(fd2);
  dup2(fd1, 0); dup2(fd2, 1);
fd1 must be duplicated to fd 0, and fd2 be 1.

Guide for grading - give 1/1/3 points for each black.

6.3 (10 pt) Refer to the following p1.c and p2.c. Assume that there are two programs, p1 (executable program of p1.c) and p2 (executable program of p2.c) in the same directory. How many 'A's would be printed if we run p1 with argument "4" (i.e. "`./p1 4`")?

```
// p1.c

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char** argv) {
  int n = atoi(argv[1]);     // convert argument into integer
  if (n < 1 || n > 9) return 0;

  char buf[2] = {'0', '\0'};

  if (fork() == 0) {
    buf[0] = buf[0]+(n-1);  // buf[0] becomes digit '(n-1)'
  }
  else if (n >= 2) {
    buf[0] = buf[0]+(n-2);
  }

  char *argv_new[] = {"./p2", buf, NULL};
  execvp("./p2", argv_new);
  return 0;
}
```

```
// p2.c

#include <stdio.h>
#include <unistd.h>

int main(int argc, char** argv) {
  fprintf(stderr, "A\n");
  fork();

  char *argv_new[] = {"./p1", argv[1], NULL};
  execvp("./p1", argv_new);
  return 0;
}
```

Answer: 50
Explanation
Let A(n) be the number of printed 'A's when running "./p1 n".
A(0) = 0
A(1) = 2 (Both parent and child execute "./p2 0")
When n >= 2,
A(n) = (1 + 2*A(n-1)) + (1 + 2*A(n-2)) = 2*(1 + A(n-1) + A(n-2))
Thus, A(2) = 6, A(3) = 18, A(4) = 50

Guide for grading - give 3 points for 34 (1 error that A(1) = 1)

Student ID: Name:

7. (10 pt) Signal

7.1 (4 pt) Explain the sequence of events when a user types `Ctrl-C` in a Unix-based system. Your explanation should cover the following:

- What signal is generated.
- The role of the operating system in handling this signal.
- The default action associated with this signal.

Answer:

When a user types `Ctrl-C`, the following sequence of events occurs:

- The keystroke generates an interrupt.
- The operating system handles this interrupt by sending a `SIGINT` signal to the application process running in the foreground.
- The default action associated with the `SIGINT` signal is to terminate the process.

grading criteria

- What signal is generated. (+1)
- The role of the operating system in handling this signal. (+1)
- The default action associated with this signal. (+2)

7.2 (2 pt) Explain the differences between using `raise()` and `kill()` functions to send signals in Unix-based systems.

Answer:

`raise()`: Sends a signal to the calling process itself. (+1pt)

`kill()`: Sends a signal to any process specified by its PID. (+1pt)

7.3 (4 pt) Modify the following skeleton code to block the `SIGINT` signal while performing a critical section of code and then unblock it afterwards. Ensure the program handles the `SIGINT` signal by printing "SIGINT received" when it is caught.

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

void sigint_handler(int sig) {
    printf("SIGINT received\n");
}

int main() {
    sigset_t set;

    // Install the signal handler
    // (student code here)
    ____________(A)_____________

    // Initialize the signal set
    sigemptyset(&set);
    sigaddset(&set, SIGINT);

    // Block SIGINT
    // (student code here)
    ____________(B)_____________

    // Critical section
    printf("In the critical section\n");
    sleep(5); // Simulate critical section work

    // Unblock SIGINT
    // (student code here)
    ____________(C)_____________

    // Infinite loop to keep the program running
    while (1) {
        // Waiting for signals
    }

    return 0;
}
```

Answer:

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

void sigint_handler(int sig) {
    printf("SIGINT received\n");
}

int main() {
    sigset_t set;

    // Install the signal handler
    signal(SIGINT, sigint_handler);

    // Initialize the signal set
    sigemptyset(&set);
    sigaddset(&set, SIGINT);

    // Block SIGINT
    sigprocmask(SIG_BLOCK, &set, NULL);

    // Critical section
    printf("In the critical section\n");
    sleep(5); // Simulate critical section work

    // Unblock SIGINT
    sigprocmask(SIG_UNBLOCK, &set, NULL);

    // Infinite loop to keep the program running
    while (1) {
        // Waiting for signals
    }

    return 0;
}
```

grading criteria
+2 for the first correct line of code
+1 for each second/third correct line of code

Student ID: Name:

8. (10 pt) Short answers

8.1 (2 pt) During the execution of a program, do you expect to have a higher cache miss rate or a higher page "miss" rate (or page faults)? Please explain in one sentence.

Answer: higher cache miss rate. page granularity is much larger than a cache line.

grading criteria

- no partial point

8.2 (2 pt) Provide one similarity and one difference between exceptions and function calls.

Answer:
similarities: both result in a change of control flow.
difference: there can be many different answers.

- exceptions sometimes do not return, function calls always return.
- exceptions occur based on some events (or traps) while functions are initiated by the programmer
- exceptions can require support from the kernel while functions are all within the user code.

grading criteria

- similar (1pt)
- difference (1pt)

8.3 (4 pt) Page Table
(a) (1 pt) Assume that a page table entry is 4B (ignore the valid bit) and there are 16k virtual pages. How big is the page table?

Answer: 4B x 16k = 64kB

grading criteria: no partial point

(b) (1 pt) Now assume that there are 20 processes in the system. How does the page table size change to support 20 processes?

Answer: per-process page table size does not change (64kB) but the each process needs its own page table —> 64kB x 20 = 1280 KB

grading criteria: no partial point.

(c) (2 pt) Page tables are often stored in memory because of their size. However, this incurs latency overhead since all memory accesses require two accesses (one for the page table and one for the actual data itself). To accelerate the translation, a dedicated (small) hardware cache is proposed that saves recently translated information. Is this a good idea? Why or why not?

Answer: Yes because of locality. Recently used translation will likely be used again.

grading criteria
- 1pt : if student says yes

8.4 (2 pt) Which instructions can potentially cause an exception and result in the same instruction being re-executed?

(a) DIV (divide) (b) JE (jump) (c) MOV (d) ADD

Answer: (C) page fault caused by memory load instruction
(B) also can be an answer. When the program size is large, the operating system loads the portion of the program on the memory, to save the memory space. So when the program jumps to the portion of program that is not in memory, the page fault can occur.

grading criteria
- select b, c -> 2pts
- select only b -> 2pts
- select only c -> 2tps
- select b or c with a or d -> 1pt
- did not select b nor c : 0pt