System software 期末考 **2024/12/12**

總分 108 分

(1) To enhance the efficiency of library searching, a specific data structure is often used. Please describe its structure and functionality in detail. (5 pts)
(2) Use a diagram to illustrate the processing of an object program through a linking loader and a linkage editor. Highlight the differences between these two mechanisms. (5 pts)
(3) Why is it important for the code in a dynamic linking library to be position-independent? (5 pts)
(4) What challenges arise when a macro definition contains labels? Provide potential solutions. (5 pts)
(5) Explain the concept of conditional macro expansion and describe its applications. (5 pts)
(6) Compare positional parameters and keyword parameters in the context of macro processing. (5 pts)
(7) What occurs when a system call is invoked? (3 pts)
(8) What happens if a system call encounters an error? How can you write a statement to explain the cause of the error? (7 pts)
(9) What are the differences between system calls and library calls? (5 pts)
(10) What issues might arise if an operating system relies solely on library calls without providing system calls? (5 pts)
(11) Define a file descriptor in UNIX. What is the relationship between the maximum number of files a process can open and the file descriptor? (5 pts)
(12) Describe the fork() system call in UNIX. How does it differ from the exec() system call? (5 pts)
(13) Pipes and message queues are two mechanisms for interprocess communication in UNIX. Compare their differences. (8 pts)
(14) What are the limitations of using pipes for interprocess communication in UNIX? Why does the pipe() system call in UNIX return two file descriptors? (5 pts)
(15) What is a “SIGNAL” in UNIX? What information is contained in the arguments of the signal() system call? (5 pts)
(16) You need to design a program on a UNIX system to manage access permissions for a file (note that this file cannot be split). The file contains multiple rows, and each row needs to be accessible by different individuals. How would you design this program? Provide only the implementation approach without writing the actual code. (10 pts)
(17) Please implement a program using Unix system calls to solve the following problem: You have an executable interactive program that communicates with the user via standard input and output. It receives commands from standard input, processes them, and sends the results to standard output. However, you do not have the source code for this program; you only have its executable image (a file similar to a.out named xyz). Now, you want to use the functionality of this program through another program. Your designed program must send commands to this executable and retrieve the results of those commands, displaying them accordingly. .
    1. First, illustrate the architecture of your solution, including the data flow and required components. Indicate the I/O redirection clearly in the diagram. (5 pts)
    2. Then, write the C code for this implementation. (15 pts)

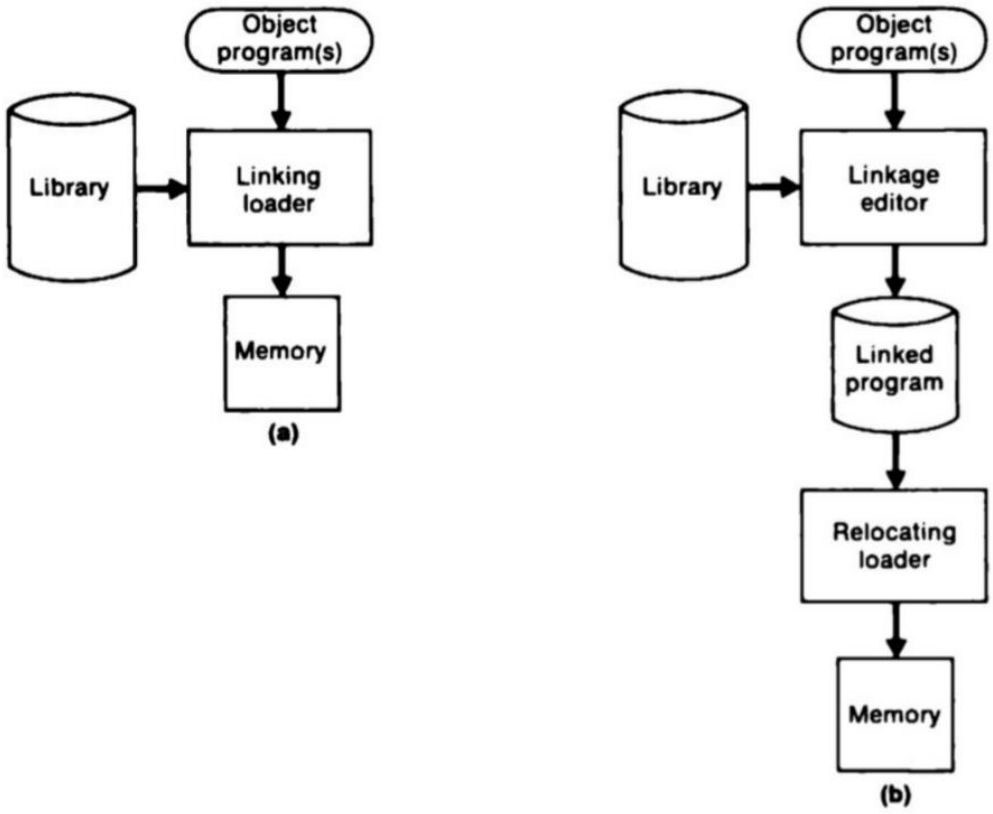
1. Directory,一個對應 symbol name 和其在何 object program 的 table，其中 symbol name 有排序過，以支援快速 search。(5pts)
2. 

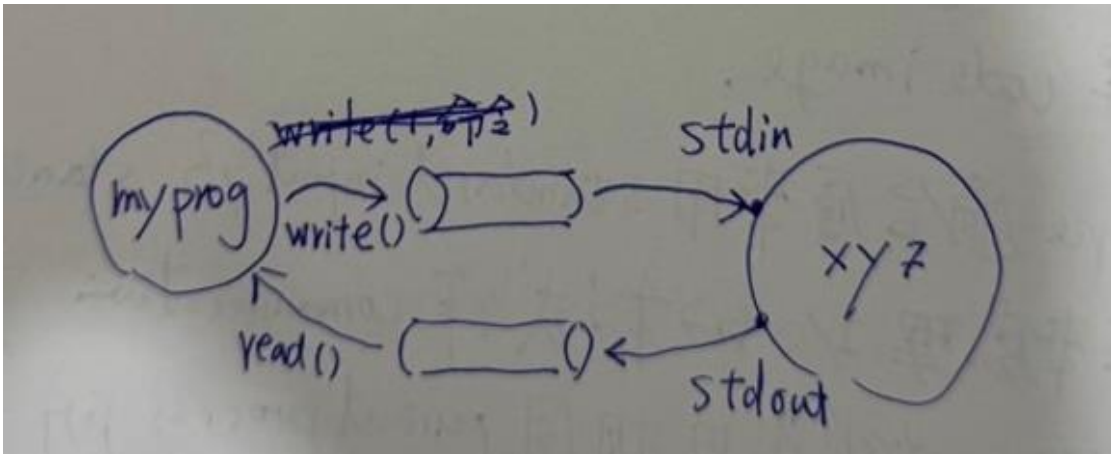
Figure 3.13 Processing of an object program using (a) linking loader and (b) linkage editor.

Differences: Linking loader 每次執行程式都要作一次 linking,這必須作一次 library search。 Linking editor 將 object programs 合併在 linked program(executable image)，以後執行不用 linking。(5pts)

3. 因為這些 code 被許多不同之 process 共享，其在這些 process 的記憶空間之位置不固定。(5pts)
4. (1)兩次 macro expansion 會產生相同之 label。(2)Allowing the creation of special types of label within macro instructions:如 loop (5pts)
5. Most macro processors can modify the sequence of statements generated on the arguments supplied in the macro invocation.(大多數的巨集處理器可以根據巨集調用中提供的參數，修改所生成語句的順序。) (5pts)
6. P: Parameters and arguments were associated with each other according to their positions in the macro prototype and the macro invocation statement. Consecutive commas is necessary for a null argument.(參數和引數根據它們在巨集原型與巨集調用語句中的位置進行關聯。連續的逗號對於表示空引數是必要的。)

K: Each argument value is written with a keyword that names the corresponding parameter. A macro may have a large number of parameters, and only a few of these are given values in a typical invocation.(每個引數值都使用關鍵字來標示對應的參數名稱。一個巨集可能具有大量的參數，但在典型的調用中，只有少數參數會被賦予值。) (5pts)

7. 產生 software interrupt(trap) 進入 kernel mode (3pts)
8. (1)Return -1, The reason of tailing in an external variable “errno”. (2)可以呼叫 perror() system call. (7pts)
9. System call 會產生 software interrupt (5pts)
10. 無法作 resource 的 protection. (5pts)
11. File descriptor 是在一個 process 的 PCB(process control block)中的一個 array of pointers.
    其 array 的 size 就是此 process 同時最多可以地開檔數目。 (5pts)
12. Fork()會將 parent process 的 code image(包刮 text,data,heap,stack)及 PCB 中大部分的 data 都 duplicate 一份成為 child process 的 code image 及 code image. 只有回傳參數不同:parent process 的 fork()回傳 child process 的 pid: child process 的 fork()回傳 0.
    Exec()由一個 executable image(在 file system)重建 code image. (5pts)
13. Pipe 適合原來用 standard input 及 standard output 來處理 I/O 的程式作 communication, 這些 process 必須是由相同 parent process 所 fork()。
    Message queue: 利用一個約定的 key 來建立一個 queue。通訊的 process 間可以沒有特殊的關係(parent, child, sibling) (8pts)
14. (1) 通訊的兩個 processes 必須是同一個 parent 所 fork 出來，或是有 parent-child 的關係。
    (2) 因為一個 pipe 有讀取端及寫入端，UNIX 把這兩端當成檔案一樣用 read()及 write()來讀取及寫入資料，所以會回傳兩個 file descriptors。(5pts)
15. 將一個 process 掛上 signal handling routine。
    Handling routine 的 pointer 及 signal 的號碼。(5pts)
16. 此程式必須設定成一個 Setuid program。
    (chmod u+s “prog”)
    此程式的 owner 必須能 access 此 file。
    其他 user 必須設定成不能 access 此 file。
    此程式在執行時必須一開始確定執行者之身分(比如擁有帳號+密碼)，然後根據執行者之身分取出被保護 file 中之資料。(10pts)
17. (1)建立兩個 pipe, fork() xyz 並對 xyz 的 standard I/O 轉向 (5pts)



(2)(15pts)

```
1.  #include<stdio.h>
2.  #include<stdlib.h>
3.  #include<sys/types.h>
4.  #include<sys/wait.h>
5.  #include<unistd.h>
6.  #include<string.h>
7.
8.  int main()
9.  {
10.     int pid, pid1, status;
11.     int pfd[2], pfd2[2]; /* pipe file descriptor table. */
12.
13.     /*  建立第一組  pipe */
14.     if (pipe(pfd) == -1)    {
15.         perror("pipe pfd failed");
16.         exit(1);
17.     }
18.
19.     /*  建立第二組  pipe */
20.     if (pipe(pfd2) == -1) {
21.         perror("pipe pfd2 failed");
22.         exit(1);
23.     }
24.
25.     /* fork  建立子行程  */
26.     pid = fork();
27.     switch(pid) {
```

```
28.             case -1:
29.                 /* fork 失敗的情況 */
30.                 printf("The system can not create process now.\n");
31.                 return 0;
32.
33.             case 0:
34.                 /* 子行程 (Child process) */
35.
36.                 /* 將標準輸入重導向為 pfd[0] */
37.                 close(0);
38.                 dup(pfd[0]);
39.                 close(pfd[0]);
40.                 close(pfd[1]); // 子行程不用寫入 pfd
41.
42.                 /* 將標準輸出重導向為 pfd2[1] */
43.                 close(1);
44.                 dup(pfd2[1]);
45.                 close(pfd2[1]);
46.                 close(pfd2[0]); // 子行程不用從 pfd2 讀取
47.
48.                 /* 在執行 execl 前印出子行程訊息，這行會透過 pfd2 管道傳
    到父行程 */
49.                 printf("I am the child process\n");
50.
51.                 /* 執行 wc 程式 */
52.                 execl("/usr/bin/wc", "wc", (char *) NULL);
53.                 perror("execl failed");
54.                 exit(1);
55.
56.             default:
57.                 /* 父行程 (Parent process) */
58.                 printf("I am the parent process\n");
59.
60.                 /* 向子行程的標準輸入 (pfd[1]) 寫入資料 */
61.                 write(pfd[1], "This is a book\n", strlen("This is a book\n"));
62.                 write(pfd[1], "That is a pencil\n", strlen("That is a pencil\n"));
63.
64.                 /* 關閉 pfd[0], pfd[1] 不再需要 */
```

```
65.                 close(pfd[0]);
66.                 close(pfd[1]);
67.
68.                 /* 父行程不需要往 pfd2 寫入，因此關閉 pfd2[1]，只保留
    pfd2[0] 用來讀取子行程輸出 */
69.                 close(pfd2[1]);
70.
71.                 /* 從 pfd2[0] 讀取子行程(wc)輸出的內容並印出 */
72.                 {
73.                     char buf[1024];
74.                     ssize_t n;
75.                     while ((n = read(pfd2[0], buf, sizeof(buf)-1)) > 0) {
76.                         buf[n] = '\0';
77.                         printf("From child output:\n%s", buf);
78.                     }
79.                 }
80.                 close(pfd2[0]);
81.
82.                 /* 等待子行程結束 */
83.                 while ((pid1 = wait(&status)) != -1) {
84.                     printf("Child process %d terminates\n", pid1);
85.                 }
86.
87.                 printf("I am the parent process, I exit\n");
88.         }
89.
90.         return 0;
91. }
```