Design Document

**作者or組員：**陳育霖 B113040052

**目標：Synchronization and Mailbox**

**新增或修改的檔案：**

System Call:misc.c proto.h table.c callnr.h

Test: shell.c sender.c receiver.c shared\_memory.h (minix版本使用)

**程式使用的函式：**

putchar 將輸入的文字放入buffer中；getchar 將buffer中的文字輸出

fork 在Spawn中負責建立新Process；waitpid 在Waitpid中用以等待子Process

execvp 在Spawn中用來執行指令；usleep 避免導致顯示問題或Race condition

strtok 將文字分割；open 開啟檔案；mmap 將地址map進變數中

shmget 建立共享記憶體；shmat 啟動記憶體功能；shmdt 關閉記憶體功能

右下方函式 -> 用以輸出buffer內容： va\_list 宣告可變參數；

va\_start 初始化可變參數；va\_arg 從中獲取下個值；va\_end 結束可變參數

**檔案路徑如下：**

misc.c /usr/src/minix/servers/pm/ 設定system call的function

proto.h /usr/src/minix/servers/pm/ 設定system call function 的格式

table.c /usr/src/minix/servers/pm/ 設定system call代碼對應的function

callnr.h /usr/src/minix/include/minix/ 設定system call的代碼

**編譯步驟如下：**

編譯system call

cd /usr/src/minix/servers/pm

make clean

make

編譯環境

cd /usr/src/releasetools

make clean

make hdboot

**流程圖：**

一張含有 文字, 螢幕擷取畫面, 字型, 行 的圖片

自動產生的描述mmap版本

一張含有 文字, 螢幕擷取畫面, 字型, 數字 的圖片

自動產生的描述圖1 傳送訊息流程圖

圖2 狀態流程圖

一張含有 文字, 螢幕擷取畫面, 字型, 行 的圖片

自動產生的描述shm版本

一張含有 文字, 螢幕擷取畫面, 字型, 行 的圖片

自動產生的描述圖3 傳送訊息流程圖

圖4 狀態流程圖

**使用的Trap功能**

Spawn 創立子Process並執行使用者輸入的指令

Waitpid 等待子Process執行結束，並在結束後return PID

Getchar 處理使用者輸入的資料，並將其存在buffer中

Putchar 將buffer中的內容輸出

ProcessFork 程式可以透過ProcessFork來建立Process

**功能介紹：**

Mailbox是單向的，一個Process負責發送訊息，其他Process負責接收訊息。

**製作過程面臨的問題：**

1. 在設計mailbox的記憶體時，直接使用mmap的話，在同個程式不同Process時，可以共用記憶體內容，但換成不同程式的不同Process時，每個程式無法知道其他程式的記憶體位置。
2. 沒有使用互斥鎖的話，接收方會一直輸出內容，使畫面混亂；而發送方會不知道接收方是否收到訊息，以及每個Process是否收到後都輸出完成。
3. 在設計shell時，putchar無法將變數加在字串中，例如：("Process%d", ID)。

**處理同步問題：**

mmap版本

為了使不同程式之間可以互相溝通，我使用.bin檔(二進位檔案)來當作一個訊息傳遞的媒介，將mmap的位置指向該檔案，進而達到共享記憶體內容的功能，並使用一個bin檔來傳送lock的資訊，同時解決同步問題。

shm版本

在shared\_memory.h中設定key，不同的程式使用相同的key來設定記憶體，這樣程式間就可以使用相同記憶體位置來傳送資訊，並在該記憶體中設定lock變數來處理同步問題。

**預防重複讀取：**

使用lock來記錄。當Process接收到消息，並順利輸出後，lock的數字會從0變成1，這樣迴圈再次跑到判斷時，就會鎖住，避免重複讀取與輸出。當全部訊息都輸出後，發送訊息的Process的鎖被解開時，會提示使用者輸入數字，當使用者輸入完數字後，lock的數字會從1變成0，此時，接收訊息的Process才可再次輸出新的內容。

**測試結果：**

**Input & output** 輸入: lab^H^Hscd^H^H -t // <CONTROL-H>是Backspace

**一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述**

圖5 輸入輸出測試

**Testing** (test1為發送者、test2為接收者)

接收者只會輸出，不須輸入，因此讓其在後台執行；而發送者需一直輸入，因此需使用Waitpid等待。

一張含有 文字, 螢幕擷取畫面, 字型, 功能表 的圖片

自動產生的描述一張含有 文字, 字型, 螢幕擷取畫面, 設計 的圖片

自動產生的描述

圖6 Mailbox測試(2 child Process)

**檔案修改的內容(System call):**

callnr.h

#define PM\_SPAWN (PM\_BASE + 48)

#define PM\_WAITPID (PM\_BASE + 49) 圖7 Mailbox測試(4 child Process)

#define PM\_GETCHAR (PM\_BASE + 50)

#define PM\_PUTCHAR (PM\_BASE + 51)

#define PM\_PROFORK (PM\_BASE + 52)

#define NR\_PM\_CALLS 53

table.c

CALL(PM\_SPAWN) = do\_spawn, // 在最後一行加上

CALL(PM\_WAITPID) = do\_waitpid,

CALL(PM\_GETCHAR) = do\_getchar,

CALL(PM\_PUTCHAR) = do\_putchar,

CALL(PM\_PROFORK) = do\_processfork

proto.h

/\* my syscall lib \*/ // 在最後一行加上\

int do\_spawn(char \*prog, char \*arg[], int background);

int do\_waitpid (pid\_t pid);

void do\_putchar(char c);

void do\_putchar(char c);

int do\_processfork();

misc.c

void do\_putchar (char c) {

putchar(c);

}

char input\_buffer[100]; // getchar

int buffer\_index = 0; // getchar

void do\_getchar(char c) {

if (c == '\n') {

input\_buffer[buffer\_index] = '\0';

buffer\_index = 0;

} else if (c == '\b') {

Backspace();

} else {

if (buffer\_index < BUFFER\_SIZE - 1) {

input\_buffer[buffer\_index++] = c;

}

}

}

int do\_waitpid(pid\_t pid) { // Waitpid函式

int status;

pid\_t ret\_pid = waitpid(pid, &status, 0); // 等待子Process結束

if (ret\_pid == -1) {

perror("waitpid");

return -1; // 表示有問題

}

if (WIFEXITED(status)) {

printf("Child process %d exited with status %d\n", ret\_pid, WEXITSTATUS(status));

} else {

printf("Child process %d did not exit normally\n", ret\_pid);

}

return ret\_pid; // return Process的ID

}

int do\_processfork() { // 建立Process的函式

pid\_t pid = fork(); // 調用 fork 函式創建一個新的Process

if (pid == -1) {

return -2; // return 錯誤

} else if (pid == 0) {

// 子進程

//printf("Child process created with PID: %d\n", getpid());

} else {

return pid; // return Process的ID

}

return pid;

}

int do\_spawn(char \*prog, char \*arg[], int background) { // 執行指令及分配Process

int result = ProcessFork(); // 調用 ProcessFork 函式創建一個新的Process

if (result < 0) {

return result; // 如果創建失敗，直接返回錯誤碼

} else if(result == 0) {

execvp(prog, arg);

perror("execvp");

}

if(!background) // 在背景執行，不須等待

return Waitpid(result); // return Process的ID

usleep(50);

return result;

}