作業系統工程期末project

李奕承 611121212

我選擇做的是timer(alarm)

先前已經先做了
1.將這學期寫的每個作業組合起來
2.將資料結構課寫的 double_linked_list.a 移植進來
3.實現了顯示所有 task PCB 及系統時間的簡單桌面
(建議老師直接編譯執行，可以看到整個系統的變化)

PCB結構

```
typedef struct pcb{
    dllNode_t node;
    context context;
    uint8_t stack[STACK_SIZE];
    int ID;
    int priority;
    int finish;
    int waiting;
    int error;
    reg_t pasue_address;
}pcb_t;
```

當我新增一個 task 時，會 malloc 一個 pcb_t 結構加入 task 清單串列，並設定好必要的值

```
int task_create(void (*start_routin)(char *param), char *param, uint8_t priority)
{
    pcb_t * porcess = (pcb_t *)malloc(sizeof(pcb_t));
    if (porcess != NULL) {
        porcess->ID = process_ID;
        porcess->finish = 0;
        porcess->waiting = 0;
        porcess->node.next = NULL;
        porcess->node.prev = NULL;
        porcess->priority = priority;
        porcess->pasue_address = (reg_t) start_routin;
        porcess->context.sp = (reg_t) &(porcess->stack[STACK_SIZE -1]);
        porcess->context.ra = (reg_t) start_routin;
        porcess->context.a0 = (reg_t) param;
        process_ID++;
        DLL_add_tail(&porcess->node, pcb_list);
```

```
        return 0;
    } else {
        return -1;
    }
}
```

在中斷處理結束前會使用一次 showPCB() 來刷新畫面，畫面中包含

- 畫出一個PCB表格
- 運作中的timer(alarm)
- 系統時間
- 還有使用者輸入過的文字暫存

```
#include"os.h"


void showPCB()
{
    int task_count = DLL_num_nodes(pcb_list);
    printf("\033[?25h");
    uart_puts("\E[H\E[J");
    dllNode_t * target = pcb_list;
    printf("\n| task ID  ");
    while(target->next != NULL)
    {
        target = DLL_next_node(target);
        printf("|   %d    ", ((pcb_t *)target)->ID);

    }
    printf("|\n");
    printf("------------");
    for(int i = 0 ; i < task_count ; i++)
    {
        printf("---------");
    }
    printf("\n| priority ");
    target = pcb_list;
    while(target->next != NULL)
    {
        target = DLL_next_node(target);
        //顯示所有task的優先級
        printf("|   %d    ", ((pcb_t *)target)->priority);
    }
    printf("|\n");
    printf("------------");
    for(int i = 0 ; i < task_count ; i++)
    {
        printf("---------");
    }
    printf("\n|  finish  ");
```

```
    target = pcb_list;
    while(target->next != NULL)
    {
        target = DLL_next_node(target);
        if(((pcb_t *)target)->finish == 1)
        {
            printf("|\033[1;31;1m   1    \033[0;37;1m");
        }
        else
        {
        //顯示所有task是否被執行過的狀態
        printf("|   0    ");
        }
    }
    printf("|\n");
    printf("------------");
    for(int i = 0 ; i < task_count ; i++)
    {
        printf("---------");
    }

    printf("\n|  waiting ");
    target = pcb_list;
    while(target->next != NULL)
    {
        target = DLL_next_node(target);
        if(((pcb_t *)target)->waiting == 1)
        {
            printf("|\033[1;31;1m   1    \033[0;37;1m");
        }
        else
        {
        //顯示所有task是否被執行過的狀態
        printf("|   0    ");
        }
    }
    printf("|\n");
    printf("------------");
    for(int i = 0 ; i < task_count ; i++)
    {
        printf("---------");
    }

    printf("\n|  error   ");
    target = pcb_list;
    while(target->next != NULL)
    {
        target = DLL_next_node(target);
        if(((pcb_t *)target)->error == 1)
        {
            printf("|\033[1;31;1m   1    \033[0;37;1m");
        }
        else
```

```
        {
        //顯示所有task是否被執行過的狀態
        printf("|    0    ");
        }
    }
    printf("|\n\n");
    show_alarm();
    clock();
    printf(":%s", trap_temp);
}
```

畫面長這樣

以下是我為了測試所產生的9個 task ，他們每個會輸出不一樣的圖形。 其中第 3、6、8 task 會在運行時申請一個鬧鐘，並進入等待狀態。
task 2 會故意產生一個異常(上圖可以看到)。

```
//第1個task
void user_task1(char* santams)
{
    printf("Task 1: Running...|%s\n", santams);
    uart_puts("                  |...\n");
    uart_puts("                     |...\n");
    uart_puts("                        |...\n");
    asm volatile("wfi");
    task_exit();
}
//第2個task
void user_task2(char* santams)
{
    printf("Task 2: Running...........|.........%s\n", santams);
    uart_puts("                           ...|\n");
    uart_puts("                        ...|\n");
    uart_puts("                     ...|\n");
    trap_test();
    asm volatile("wfi");
    task_exit();
}
```

```c
//第3個task
void user_task3(char* santams)
{
    printf("                    Task 3: Running...|%s\n", santams);
    uart_puts("                                      |...\n");
    uart_puts("                                         |...\n");
    uart_puts("                                            |...\n");
    create_alarm(20);
    task_wait();
    asm volatile("wfi");
    task_exit();
}
//第4個task
void user_task4(char* santams)
{
    printf("                    Task 4: Running...........|.........%s\n", santams);
    uart_puts("                                           ...|\n");
    uart_puts("                                         ...|\n");
    uart_puts("                                       ...|\n");
    asm volatile("wfi");
    task_exit();
}

//第5個task
void user_task5(char* santams)
{
    printf("                                           Task 5: Running...|%s\n", santams);
    uart_puts("                                                         |...\n");
    uart_puts("                                                            |...\n");
    uart_puts("                                                               |...\n");
    asm volatile("wfi");
    task_exit();
}
//第6個task
void user_task6(char* santams)
{
    printf("                                           Task 6: Running...........|.........%s\n"
    uart_puts("                                                                  ...|\n");
    uart_puts("                                                               ...|\n");
    uart_puts("                                                            ...|\n");
    create_alarm(10);
    task_wait();
    asm volatile("wfi");
    task_exit();
}

//第7個task
void user_task7(char* santams)
{
    printf("                                                                  Task 7: Running...|%s\
    uart_puts("                                                                                 |...
    uart_puts("                                                                                    |
    uart_puts("
```

```
    asm volatile("wfi");
    task_exit();
}
//第8個task
void user_task8(char* santams)
{
    printf("                                                    Task 8: Running.......
    uart_puts("
    uart_puts("                                                                      .
    uart_puts("                                                                    ...|
    create_alarm(40);
    task_wait();
    asm volatile("wfi");
    task_exit();
}

//第9個task
void user_task9(char* santams)
{
    printf("                                                                        Ta
    uart_puts("
    uart_puts("
    uart_puts("
    asm volatile("wfi");
    task_exit();
}
```

alarm_t 結構

```
typedef struct alarm{
    dllNode_t node;
    int tickets;
    int ID;
    dllNode_t * owner;
}alarm_t;
```

當要產生一個鬧鐘時，會 malloc 一個 alarm_t 結構加入 alarm 清單串列，並設定好必要的值，接著判斷要插在哪個時鐘前面，插入後減掉在這之前所有鬧鐘加總的 tickets，接著我的下一個鬧鐘也要減掉我的 tickets

```
void create_alarm(int tickets)
{
    alarm_t * new = (alarm_t *)malloc(sizeof(alarm_t));
    new->node.next = NULL;
    new->node.prev = NULL;
    new->owner = current_task;
    new->tickets = tickets;
    new->ID = ((pcb_t *)current_task)->ID;
```

```
    dllNode_t * temp = alarm_list;
    int before = 0;
    int after = 0;
    while (1)
    {
        if(temp->next == NULL)
        {
            DLL_addto_next(&new->node, temp);
            new->tickets -= after;
            return;
        }
        else
        {
            temp = temp->next;
            before += after;
            after += ((alarm_t *)temp)->tickets;
            if(new->tickets < after)
            {
                DLL_addto_prev(&new->node, temp);
                new->tickets -= before;
                ((alarm_t *)temp)->tickets -= new->tickets;
                return;
            }
        }
    }
}
```

每當一個時鐘中斷來時，除了更新系統時間，也會把 alarm 清單串列的第一個鬧鐘減掉 1 ticket ，如果減為0，就把自己從串列中刪除，並把當初申請鬧鐘的 task 喚醒

```
void alarm_handle()
{
    dllNode_t * temp = DLL_next_node(alarm_list);
    if(temp != NULL)
    {
        ((alarm_t *)temp)->tickets--;
        if(((alarm_t *)temp)->tickets == 0)
        {
            ((pcb_t *)(((alarm_t *)temp)->owner))->waiting = 0;
            DLL_delete(temp);
            free(temp);
        }
    }
}
```

alarm_handle() 放在中斷處理程式中 timer_handle()的下面

```
reg_t trap_handler(reg_t epc, reg_t cause)
{
    ((pcb_t *)current_task)->pasue_address = epc;
    reg_t return_pc = epc;
    reg_t cause_code = cause & 0xfff;

    if(cause & 0x80000000)
    {
        switch (cause_code)
        {
        case 3:
            //uart_puts("software interruption!\n");
            {
            int id = r_mhartid();
            *(uint32_t *)CLINT_MSIP(id) = 0;
            //((pcb_t *)current_task)->context.ra = return_pc;
            showPCB();
            //schedule();
            current_task = &(manage->node);
            return (reg_t)schedule;
            }
            break;

        case 7:
            //uart_puts("timer interruption!\n");
            timer_handle();
            alarm_handle();
            break;

        case 11:
            //uart_puts("external interruption!\n");
            external_interrupt_handler();
            break;


        default:
            uart_puts("unknown async exception!\n");
            break;
        }
    }
    else
    {
        printf("Sync exceptions!, code = %d\n", cause_code);
        uart_puts("OOPS! What can I do!\n");
        uart_puts("next time, this task will start at pc where exeception next \n");
        ((pcb_t *)current_task)->error = 1;
        //return return_pc += 4;
        return (reg_t)schedule;
    }
    showPCB();
```

```
    return return_pc;
}
```

以下是我的 schedule() 他會掃描整個 task 清單串列，尋找可以執行的task並選出 priority 最大的，然後跳進去執行。如果只剩下 ID 為 0 的task(schedule本身的ID)，代表目前沒有其他事可做，所以會開始回收已經被標註為 finish 或者 error 的 task，將其pcb_t結構從清單中刪除並釋放。

```
void schedule()
{
    int task_count = DLL_num_nodes(pcb_list);
    dllNode_t * max = &(manage->node);
    dllNode_t * target = pcb_list;
    int temp = 0;
    while(target->next != NULL)
    {
        target = DLL_next_node(target);
        //找到優先度最高的task，並且必須是沒有被做過的，且沒有在等待中
        if(((pcb_t *)target)->priority >= temp && ((pcb_t *)target)->finish != 1 && ((pcb_t *)
        {
            //將最優先要做的task編號設給 max 變數
            temp = ((pcb_t *)target)->priority;
            max = target;
        }
    }


    //如果 ((pcb_t *)max)->priority 為 0 且 ((pcb_t *)max)->finish 也是 0，代表系統目前沒有其他任
    if(((pcb_t *)max)->priority == 0 && ((pcb_t *)max)->finish == 0)
    {
        current_task = max;

        target = pcb_list;
        while(target->next != NULL)
        {
            target = DLL_next_node(target);
            if(((pcb_t *)target)->error == 1 || ((pcb_t *)target)->finish == 1)
            {
                DLL_delete(target);
                free(target);
            }
        }


        asm volatile("wfi");
    }
    //不是的話代表還有task沒做完，於是將下個 task 的 context 指針設為 max 所代表的task 然後switch_t
    else
    {
        printf("max = task %d\n\n\n", ((pcb_t *)max)->ID);
        struct context *next = &(((pcb_t *)max)->context);
        current_task = max;
```

```
        switch_to(next, ((pcb_t *)current_task)->pasue_address);
    }

}
```

其他

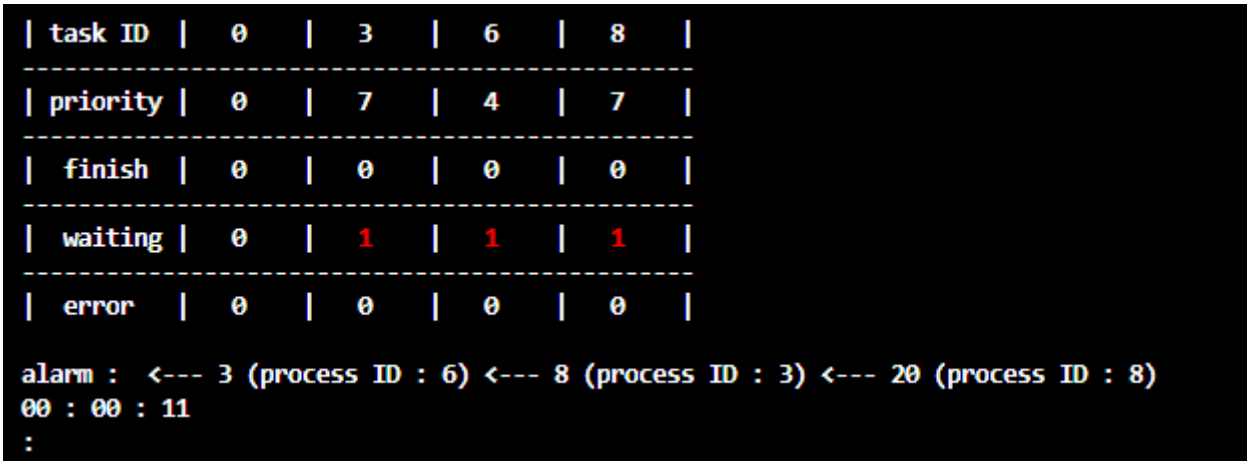
另外，如果程式是正常結束的，finish 就會被標記為 1

```
void task_exit()
{
    ((pcb_t *)current_task)->finish = 1;
    showPCB();
    switch_to( &(manage->context), manage->pasue_address);
}
```

這是我的系統時間

```
//時間進位的邏輯
void tick()
{
    seconds++;
    if(seconds == 60)
    {
        mintues++;
        seconds = 0;
        if(mintues == 60)
        {
            hours++;
            mintues = 0;
            if(hours == 24)
            {
                hours = 0;
            }
        }
    }
}
```

這是當剩下的3個 task 都在等待鬧鐘喚醒的桌面



所有task都執行完進入閒置狀態的畫面