Q1) /* ctxsw.S - ctxsw (for ARM) */

```
        .text
        .globl  ctxsw

/*------------------------------------------------------------------------
 * ctxsw -  ARM context switch; the call is ctxsw(&old_sp, &new_sp)
 *------------------------------------------------------------------------
 */

ctxsw:
        push    {r0-r11, lr}            /* Push regs 0 - 11 and lr       */
        push    {lr}                    /* Push return address           */
        mrs     r2, cpsr                /* Obtain status from coprocess.*/
        push    {r2}                    /*   and push onto stack         */
        str     sp, [r0]        /* Save old process's SP        */
        ldr     sp, [r1]        /* Pick up new process's SP   */
        pop     {r0}                    /* Use status as argument and            */
        bl      restore                 /*   call restore to restore it    */
        pop     {lr}                    /* Pick up the return address  */
        pop     {r0-r12}                /* Restore other registere       */
        mov     pc, r12                 /* Return to the new process  */
```

Let Y be the array.
Assembly code

```
//push  {r0-r11, lr}
STR r0, [Y]                             /* Place regs 0 - 11 and lr in an array*/
ADD Y,Y,#4
STR r1, [Y]
ADD Y,Y,#4
STR r2, [Y]
ADD Y,Y,#4
STR r3, [Y]
ADD Y,Y,#4
STR r4, [Y]
ADD Y,Y,#4
STR r5, [Y]
ADD Y,Y,#4
STR r6, [Y]
ADD Y,Y,#4
STR r7, [Y]
ADD Y,Y,#4
STR r8, [Y]
ADD Y,Y,#4
STR r9, [Y]
ADD Y,Y,#4
STR r10, [Y]
```

```
ADD Y,Y,#4
STR r11, [Y]
ADD Y,Y,#4
STR lr, [Y]
ADD Y,Y,#4

//push  {lr}                              /* Push return address*/
STR lr, [Y]
ADD Y,Y,#4

//mrs r2,cpsr                             /* Obtain status from coprocess.*/
mrs r2,cpsr

//push  {r2}
STR r2, [Y]
ADD Y,Y,#4

str sp, [r0]
ldr sp, [r1]

//pop   {r0}
LDR [Y2],r0
ADD [Y2],#4

bl restore                                /*cal restrore from System/intr.S*/

//pop   {lr}

LDR r0,[Y2]
ADD Y2,Y2,#4
LDR r1,[Y2]
ADD Y2,Y2,#4
LDR r2,[Y2]
ADD Y2,Y2,#4
LDR r3,[Y2]
ADD Y2,Y2,#4
LDR r4,[Y2]
ADD Y2,Y2,#4
LDR r5,[Y2]
ADD Y2,Y2,#4
LDR r6,[Y2]
ADD Y2,Y2,#4
LDR r7,[Y2]
ADD Y2,Y2,#4
LDR r8,[Y2]
ADD Y2,Y2,#4
LDR r9,[Y2]
ADD Y2,Y2,#4
LDR r10,[Y2]
```

```
ADD Y2,Y2,#4
LDR r11,[Y2]
ADD Y2,Y2,#4
LDR r12,[Y2]
ADD Y2,Y2,#4
LDR [Y],lr
ADD Y2,Y2,#4


mov     pc, r12
```

Function resume saves the resumed process’s priority in a local variable before calling ready. Show that if it references prptr->prprio after the call to ready, resume can return a priority value that the resumed process never had (not even after resumption).

Advantages:- The advantage of using an array is to avoid stack crash due to overflow.
Disadvantage:- The code becomes lengthy , complex and difficult to understand. Slower performance.

Q2)When a process kills itself, kill deallocates the stack and then calls resched, which means the process continues to use the deallocated stack. Redesign the system so a currentprocess does not deallocate its own stack, but instead moves to a new state,PR_DYING. Arrange for whatever process searches the process table to look for dying processes, free the stack, and move the entry to PR_FREE.

Ans→
When the main function of a process exits, kill is internally called.

Previously , when kill() was called , we would clear and unallocate the stack. Afte doing so , we would call the resched which would point the new process to the unallocated stack and after context switch is done, the variables of the new process is added to the unallocated stack and the current process poin

Now, In my new code , I have introduced a new state called dying. So, when the kill() is called , we change the state of the process to dying . When a process is at the dying state , resched is called before freeing the stack. So , the pointer of the old process still points to the variables of the old process in the stack. When a new process is called ,resched and context switch occurs and the new process variables are pushed in the stack and the current process pointer points to the it. After this , the stack is freed from the variables of the old process and the state is changed to free.

Q3)Function resume saves the resumed process’s priority in a local variable before calling ready. Show that if it references prptr->prprio after the call to ready, resume can return a priority value that the resumed process never had (not even after resumption).

Ans→ If prptr→prprio is called after ready , there might be a case where the resumed process may have higher priority than the currently executing process. So, when a process is put on the ready list and resched is called , an arbitrary number of processes with higher priority can execute or terminate.