Algoritmos de Servicio con entrada por teclado y control de tiempo en Mips 32: Buffer de Caracteres y Semáforo - Práctica 2

Orlando Bonilla 30869545 - Enyerber Flores 27157754

Buffer de Caracteres con Temporizador

```
.data
buffer:         .space 1024
buffer_tam:     .word 1024
cabecera:       .word 0
cola:           .word 0
tiempo:         .word 20000

fin_tiempo:     .asciiz "\nTiempo completado. Contenido del buffer:\n"
otra_linea:     .asciiz "\n"

.text
.globl main

main:
    la $s0, buffer
    lw $s1, buffer_tam
    sw $zero, cabecera
    sw $zero, cola
    lw $s4, tiempo

contar:
    li $v0, 30
    syscall
    move $t0, $a0

entrada:
    li $v0, 30
    syscall
    sub $t1, $a0, $t0
    bge $t1, $s4, imprimir_buffer
```

```
    li $v0, 12
    syscall
    blez $v0, entrada

    move $t2, $v0

    beq $t2, '\n', entrada
    beq $t2, '\r', entrada


    lw $t3, cabecera
    add $t4, $s0, $t3
    sb $t2, 0($t4)

    addi $t3, $t3, 1
    blt $t3, $s1, sin_intercambio
    li $t3, 0
sin_intercambio:
    sw $t3, cabecera

    j entrada

imprimir_buffer:
    li $v0, 4
    la $a0, fin_tiempo
    syscall

imprimir_ciclo:
    lw $t3, cola
    lw $t4, cabecera
    beq $t3, $t4, fin_imprimir

    add $t5, $s0, $t3
    lb $a0, 0($t5)

    li $v0, 11
    syscall

    addi $t3, $t3, 1
    blt $t3, $s1, int_ultimos_dos
    li $t3, 0
int_ultimos_dos:
    sw $t3, cola

    j imprimir_ciclo
```

```
fin_imprimir:
    li $v0, 4
    la $a0, otra_linea
    syscall

    j contar
```

Semáforo Temporizado

```
.data
    verde:     .asciiz "Sem foro en verde, esperando pulsador (presiona 's')\n"
    pulsado:   .asciiz "\nPulsador activado: en 20 segundos, cambiar  a amarillo
    amarillo: .asciiz "\nSem foro en amarillo, en 10 segundos cambiar  a rojo\
    rojo:      .asciiz "\nSem foro en rojo, en 30 segundos volver  a verde\n"

.text
.globl main

main:

    li $v0, 4
    la $a0, verde
    syscall

esperar_s:
    li $v0, 12
    syscall

    bne $v0, 's', esperar_s

    li $v0, 4
    la $a0, pulsado
    syscall

    li $a0, 20000
    li $v0, 32
    syscall

    li $v0, 4
    la $a0, amarillo
    syscall

    li $a0, 10000
```

```
li $v0, 32
syscall

li $v0, 4
la $a0, rojo
syscall

li $a0, 30000
li $v0, 32
syscall

j main
```

Respuestas

Funcionamiento del ciclo de una interrupción de hardware

Ejecución Normal del Programa
↓ 1. Evento de hardware
CPU detecta interrupción - Suspende ejecución - Guarda PC en EPC - Deshabilita interrupciones
↓ 2. Salto al vector
Rutina de Servicio (ISR) - Identifica dispositivo - Procesa interrupción - Limpia flag
↓ 3. Instrucción ERET
Retorno a programa - Restaura PC - Re-habilita interrupciones

Breve Explicación:

1. Un dispositivo genera una señal de interrupción (IRQ)
2. La CPU guarda el contexto actual y salta al manejador
3. Se ejecuta el código específico para atender el dispositivo
4. Al finalizar, se restaura el contexto con ERET

¿Qué diferencias hay entre gestionar E/S con sondeo y hacerlo con interrupciones?

- **Sondeo:** Es simple de implementar y tiene complejidad lineal pero consume excesivamente CPU y es muy poco eficiente, sin embargo, puede usarse sin problemas en dispositivos ultra-rápidos o cuando la CPU no tenga otras tareas.
- **Interrupciones:** Usa menos recursos y tiene baja latencia pero presenta complejidad en manejo de contextos, no obstante, en sistemas multitarea o para dispositivos lentos es muy efectivo.

¿Qué ventajas tiene el uso de interrupciones en términos de uso del procesador?

- **Maximiza el tiempo útil de CPU:** La CPU puede ejecutar código útil hasta que ocurra una interrupción, evitando ciclos perdidos en verificaciones constantes.
- **Reduce el consumo de energía:** Permite poner el procesador en estados de bajo consumo cuando no hay tareas críticas.
- **Menor sobrecarga de operaciones:** Evita accesos repetidos a registros de E/S (que son más lentos que acceder a memoria o caché).
- **Escalabilidad con múltiples dispositivos:** El hardware prioriza interrupciones (ej: temporizador >teclado), permitiendo manejar varios dispositivos sin colapsar la CPU.

¿Qué registros especiales se utilizan en MIPS32 para gestionar interrupciones?

- **Status(Registro de Estado):** Controla el estado global de interrupciones y modo de ejecución.
- **Cause(Registro de Causa):** Indica la fuente de la interrupción/excepción.
- **EPC(Contador de Programa de Excepción):** Almacena la dirección de retorno (PC) donde se debe reanudar la ejecución tras atender la interrupción.
- **BadVAddr(Dirección Inválida):** Guarda direcciones de memoria que causan excepciones (no usado en interrupciones simples).

¿Por qué es necesario guardar el contexto (registros del procesador) al entrar en una rutina de servicio?

- **Para preservar estado del programa principal:** La ISR usa registros ($t0-$t9, $a0-$a3, etc.) que podrían estar siendo utilizados por el programa interrumpido. Lo mejor sería guardar los registros en la pila antes de usarlos.
- **Para evitar corrupción de Datos:** Si la ISR modifica registros sin guardarlos, al retornar el programa principal tendría valores erróneos.

Momentos en que pueden generarse excepciones en un sistema MIPS32.

1. **4 situaciones en que puede generarse una excepción.**
   a) **Desbordamiento Aritmético (Overflow):** Operaciones con enteros que exceden el rango de 32 bits.
   b) **Fallo de Dirección (Address Error):** Acceso a memoria desalineado (ej: lw a dirección no múltiplo de 4).
   c) **Instrucción No Implementada (Reserved Instruction):** Ejecución de un código de operación no definido en MIPS32.
   d) **Interrupción Externa (Hardware Interrupt):** Puede darse por señal del teclado (como en el semáforo) o por temporizador (como los 20 segundos en el Buffer).
2. **¿Qué etapas del pipeline pueden provocar una excepción?.**
   a) **Etapa IF:** TLB Miss (fallo en Translation Lookaside Buffer), el MMU detecta que la dirección no está mapeada o carece de permisos.
   b) **Etapa ID:** Reserved Instruction, la unidad de control identifica un opcode inválido o una instrucción privilegiada ejecutada en modo usuario.
   c) **Etapa EX:** Overflow, la ALU detecta un resultado fuera de rango en operaciones con enteros.
   d) **Etapa MEM:** Address Error, la unidad de memoria verifica alineación y permisos.
   e) **Etapa WB:** Generalmente no genera excepciones ya que solo escribe en registros.

Estrategias de tratamiento de excepciones e interrupciones.

1. **Diferencia entre interrupciones y excepciones.**
   a) **Interrupciones:** Son Asíncronas (ocurren en cualquier momento, independientemente del programa). Pueden ser originadas por eventos externos al procesador (como entrada por teclado o temporizadores).
   b) **Excepciones:** Son Sincronas (provocadas directamente por la ejecución de una instrucción). Puede ocurrir debido a Overflows, accesos a memoria no válidos o intrucciones no válidas.
2. **Estrategias para tratar excepciones en un sistema.**
   a) **Reintento:** El sistema reintenta ejecutar la instrucción que causó la excepción después de corregir el error.
   b) **Terminación Controlada:** El sistema termina el proceso que causó la excepción y notifica al usuario/sistema operativo.
3. **¿Cómo se redirige la ejecución hacia la rutina de servicio?**
   a) **Evento Desencadenante (Puede ser señal de hardware o error durante la ejecución).**
   b) **Acciones Automáticas del Hardware:** Suspende la ejecución actual, guarda el contexto (almacena el PC en el registro EPC) y busca identificar la causa.
   c) **Salto al Vector de Excepciones:** El procesador redirige el flujo a una dirección fija.
   d) **Rutina de Servicio (ISR):** Esta guarda los registros importantes en la pila, luego identifica la fuente (ya sea excepción o interrupción), ejecuta el manejo específico, restaura los registros y ejecuta el EREC.
   e) **Retornar el EREC:** Restaura el PC desde EPC.

Habilitación de interrupciones en dispositivos y procesador

- **El teclado:** Se comunica mediante memoria mapeada (MMIO). Se usa la dirección 0xFFFF0000 (registro de control/status del teclado). Luego se habilitan las interrupciones en el Procesador (CP0). El procesador saltará a la dirección 0x80000180 cuando ocurra una interrupción, y ahí, identifica al teclado, lee la tecla y limpia el flag de interrupción.
- **La pantalla:** Suele ser un dispositivo mapeado en memoria, si es de solo escritura, no requiere interrupciones, pero si permite de estas cuando el buffer de pantalla está lleno.

- **El procesador:** Estos registros se habilitan configurando el registro Status del CP0, especificamente en los bits:

    1. **Bit 0(IE):** Interrupciones globales en donde 1 representa que están habilitadas y 0 que no.
    2. **Bit 1(EXL):** Nivel de excepción en donde 1 activa el modo kernel y 0 el modo usuario.
    3. **Bits 8-15(IM): Máscara con interrupciones de hardware** en donde cada bit habilita una linea de IRQ.

- **¿Qué pasaría si habilitamos interrupciones en los dispositivos, pero no en el procesador?**

    1. **Las IRQs No se Atenderán:** Los dispositivos generarán señales de interrupción (ej: al presionar una tecla o terminar un temporizador), pero el procesador las ignorará.
    2. **Posible Pérdida de Eventos:** Si un dispositivo (como un teclado básico) no almacena eventos, se perderá la tecla presionada al generar una nueva IRQ.
    3. **Uso Ineficiente de la CPU:** El programa tendrá que usar sondeo (polling) para verificar los dispositivos, desperdiciando ciclos de CPU.

Procesamiento de interrupciones.

- **¿Qué ocurre cuando se produce una interrupción de reloj?**

    1. **Evento de Interrupción (Hardware):** El temporizador (reloj) alcanza el valor programado; El dispositivo activa la línea de IRQ correspondiente.
    2. **Detección por la CPU (Hardware):** Al final del ciclo de instrucción, la CPU chequea si Status.IE = 1 (interrupciones globales habilitadas) o si Status.IM tiene el bit de la IRQ activo (ej: bit 8 para IRQ0).

       Si procede entonces suspende la ejecución actual, guarda el PC actual en EPC, establece Status.EXL = 1 (entra en modo kernel), deshabilita interrupciones (Status.IE = 0).
    3. **Salto al Vector de Interrupciones (Hardware):** La CPU salta a la dirección fija 0x80000180.
    4. **Ejecución de la Rutina de Servicio (Software - ISR).**
    5. **Retorno (Hardware + Software):** La instrucción eret restaura IE = 1 y EXL = 0 (habilita interrupciones y vuelve a modo usuario) y retorna a la dirección guardada en EPC (programa principal).

       En este proceso, el registro del PC se guarda de forma automática en el EPC, el Status se modifica, cause setea y los registros normales se guardan en la pila para evitar corrupción.

- **¿Por qué es importante guardar el contexto al entrar en la rutina?:** Para preservar registros que podrían dañarse si son usados por el programa interrumpido. También para volver a la dirección inicial y continuar en la siguiente instrucción.

Interrupciones de reloj y control de ejecución.

- **Evitar bucles infinitos:** El reloj se configura para generar una interrupción periódica (ej: cada 10 segundos). En cada interrupción, un contador de tiempo del programa se decrementa.Si el contador llega a 0, el sistema operativo (SO) asume que el programa está en un bucle infinito y lo termina.
- **Limitar Tiempo de Ejecución:** Al iniciar un programa, el SO carga un valor inicial en el temporizador (ej: 1 segundo). Si la interrupción ocurre antes de que el programa termine, el SO lo finaliza forzosamente.
- **¿Qué debe hacer el software si el programa finaliza antes de que ocurra la interrupción de reloj?:** Se puede desactivar la interrupción del reloj, o puede reiniciar el temporizador.

Análisis y Discusión de los Resultados.

- **Interrupciones vs. Sondeo:** Las interrupciones demostraron ser más eficientes, evitando el consumo innecesario de ciclos de CPU. En el buffer circular, el uso de IRQs permitió procesar datos en segundo plano sin bloquear la ejecución principal.
- **Manejo de excepciones:** La protección del contexto (EPC, registros) aseguró que el sistema recuperara correctamente su estado tras errores como overflows o accesos inválidos a memoria.
- **Interrupciones de reloj:** Permiten limitar la ejecución de programas (evitando bucles infinitos) pero requiere sincronización precisa (como deshabilitar el temporizador si el programa termina antes).
- **Falta de guardado de contexto:** Causa corrupción de datos en pruebas iniciales.