ENTRADA E SAÍDA!

Sistema de computação é formado pela UCP, MEMÓRIA e PERIFÉRICOS.
Os periféricos permitem os usuarios se comunicarem com o sistema. Periféricos podem ser de entrada, de saída ou de entrada/saída.
Teclados e impressoras: perifericos orientados a caracteres.

COMUNICAÇÃO CPU PERIFERICOS
Perifericos são lentos em relação a ucp
Comunicação via registradores
3 principais regs:
1 de dados: processador coloca dados a serem gravados (de uma operação de leitura)
2 de status: guarda se o periferico esta livre/ocupadodo
3 de controle: escreve nesse reg os comandos a serem executados pelo periferico

Os disps de in/out são mapeados em um espaço separado na memoria, a comunicação entre eles é feita nesse espaço. Com esse espaço distinto, são necessarios comandos para as operações: IN E OUT. IN é para leitura dos regs usados na comunicação ucp – periferico. OUT para escrita de dados nesses regs.

Com o In/Out mapeado na memória, para a comunicação entre o processador e o controlador do periférico, podem ser usadas todas as instruções que fazem movimentação de dados na memória.

TIPOS DE IN E OUT

IO PROGRAMADO (POOLING)
Operaçoes in out controladas pelo SO, na qual o processador fica testando o status do periferico procurando por entrada de dados. Não interrompe a cpu, porém grande parte do tempo do cpu é dedicada a esse teste.

IO COM INTERRUPÇÃO
O periférico pede uma interrupção quando uma operação de entrada de dados acaba. Existe um tratamento específico para cada interrupção. Como o periferico independe do processador, o mesmo é livre para executar outros processos.

ACESSO DIRETO A MEMORIA
A info é transferida diretamente pra memória sem intervenção do processador. Com o DMA, o processador executa outras coisas paralelamente.

INTERRUPÇÕES

1 GERADAS PELOS PROGRAMAS – implementadas por instruções especiais
2 GERADAS POR ERROS – divisao por 0, ref a memória fora de alcance
3 GERADAS PELO CLOCK
4 GERADAS PELOS PERIFERICOS – sinalização de fim de operação i/o ou condição de erro

Tratamento

O so reage a essas interrupções com rotinas de tratamentos:

Tratamento para software:
- fim da execução do programa
- o SO executa o serviço solicitado ou encaminha a perifericos I/O

Tratamentos para hardware:
Ações executadas pelo processador:
O processador acaba execução da instrução atual;
O processador testa existência de interrupção;
O processador salva estado atual;
O processador carrega contador de programa com o endereço da rotina de tratamento da interrupção;
A execução começa na rotina de tratamento da interrupção
A rotina de tratamento da interrupção executa
O contexto de execução anterior é restaurado
A execução retorna para a rotina interrompida.

SUBSISTEMAS DE IN-OUT
Permite a uma aplicação acessar os perifericos para realização de transferencia de dados. Para isso tem 2 camadas, 1 para interface com a aplicação e outra para interagir diretamente com o hardware

TRATADORES DE INTERRUPÇÕES
relacionados diretamente com as operações de entrada e saída os tratadores de interrupção possuem três níveis lógicos: tratadores de interrupção, drivers dos periféricos e Interface com os programas de aplicação.

Quando uma operação de i/o é solicitada, o sistema operacional recebe o pedido e aciona o driver que o encaminhará ao periférico específico. O processo solicitante fica bloqueado, sem disputar o processador, até que o pedido se complete.

Os tratadores de interrupção são componentes do núcleo do SO Acionados pelas interrupções.
Quando a interrupção ocorre, a rotina de tratamento executará as ações correspondentes à interrupção. O resultado de uma interrupção é que o processo bloqueado na operação será desbloqueado pelo tratador da interrupção.
Outra fonte de interrupção é o timer. Quando uma interrupção do timer ocorre, em um sistema de tempo compartilhado, muitas ações relacionadas ao tempo devem ser executadas.

No Linux, a interrupção do timer está associada ao IRQ0 e ocorre a cada 10 ms. A entrada IRQ0 contém o endereço da rotina timer_interrupt(), que executa as seguintes ações:

• Calcula o tempo de retardo entre a ocorrência da interrupção e o disparo da rotina de tratamento;
Chama a rotina do_timer_interrupt ()

A rotina do_timer_interrupt ()executa esses passos:
Atualiza a variável jiffies, que armazena o número de ticks ocorridos desde que o sistema foi disparado
Atualiza o tempo do processo do usuário
Conta o número de processos prontos para rodar e atualiza estatísticas de uso da CPU;
Verifica se o tempo do processo corrente se esgotou.
Dispara a rotina timer_bh () .

A rotina timer_bh () dispara os procedimentos responsáveis pela execução das seguintes funções:
Atualizar tempo corrente do sistema e a data do computador e calcula a carga corrente do sistema

DRIVER DE DISPOSITIVOS:

São componentes do so que interagem com os perifericos.
Para cada periférico existe um driver específico
O driver recebe um pedido do SO (q vem do user), prepara essa requisição e envia para o controle do periferico. Se ele está ocioso, a operação ocorre imediatamente, se não, o pedido é posto na fila do periférico.

No caso de ler dados do hd, o so recebe os parametros da requisição, calcula o numero do bloco e encaminha para o drive para a leitura juntamente com outros parametros. O driver constroi a requisição e ela é posta em fila. O final da operação é sinalizado pela controladora (pela interrupção)

SUBSISTEMAS DE I/O

Estão no nucleo do SO. Um software a nivel de usuario precisa pedir por função a leitura de um bloco de memoria, por exemplo.
n = read (f, &b, nbytes); (no linux)

Esta função faz parte de uma biblioteca de funções incorporada ao código do programa do usuário. a comunicação dos programas de aplicação com o subsistema de io é implementada por um conjunto de procedimentos de biblioteca, que possui a interface de cada primitiva. Fazem parte,
além da função read, as funções write, open, close, create, etc.

IO NO LINUX

Comunicação entre programa e i/o é feita pelas chamadas de sistemas de op de in/out. Cada chamada tem sesus parametros. Ex: open, close, read, etc...
chamadas de sistemas para in/out : quase todas tratadas pelo vfs (virtual file system).
O linux suporta um grande numero de perifericos, e para cada um deve haver um driver específico.
O Linux identifica o final de uma operação de entrada e saída através de uma interrupção ou fazendo pooling.

LIGADORES CARREGADORES!

LOADERS:
Programa do sistema que executa função de carga.
- Suportam relocação e ligação
- Alguns são separados entre ligadores e carregadores

FUNÇÕES DO LOADER
- Levar um programa para a memória
- Começar a sua execução

INPUT:
PROGRAMA OBJETO
- Contem instruções e dados do programa fonte
- Faz a especificação dos endereços de memória onde esses itens devem ser carregados

FUNÇÕES BÁSICAS
ALOCAÇÃO – alocar espaços na memoria para o programa

LIGAÇÃO – resolver referências simbólicas entre programas objetos
- combinar dois ou mais programas separados
-dar informações para permitir referencia entre eles
RELOCAÇÃO – Ajusta endereços dependentes para corresponderem ao endereço alocado
Modificao programa objeto de maneira que possa ser carregado em endereços diferentes da locação originalmente especificada.
CARGA- Coloca instruções de máquina e os dados na memória física

PROJETO DE CARREGADOR ABSOLUTO
Projeto sem ligação ou relocação
Opera em um passo apenas:
-verifica o header para ver se é executavel
- le os registros e move para o endereço de emmória indicado
- ao final, salta para o endereço de início de execução

MODELOS LOADER

1 COMPILE AND GO
- montador roda na memoria
- instruções vão direto pra posição de memoria alocada
-ao termino da tradução, a primeira instrução do programa traduzido é executada.

Desvantagens:
Perda de memoria (parte do montador fica indisponível)
Precisa de retradução a cada execução
Dificuldade em tratar multiplos segmentos (especialmente dados em progs !=)
Aumento de tamanho em partes do programa acarretam realocação do montador

2 BOT LOADER
Executa quando o computador é ligado
Executa o codigo na primeira posição de memória e salta para o inicio da main do SO depois da carga;

Desvantagens:
Endereço de carga deve ser especificado
Cuidado maior na sobreposição de endereços (2 funções ao mesmo end)
Dificuldade no uso de bibliotecas
Alocação e ligação feitas pelo programador
Relocação não existente
Transferencia para a memoria realizada para o carregador

CARREGADOR GERAL
Ligação, relocação e carga
Ligação de subrotinas:
Main a deseja rodar algo do subprograma b e escreve instruções de referencia para b erradas, gerando erro no montador

Externals e globals
Instrução extern seguida de simbolos indica que esse simbolo está em outro programa mas pode ser usado no presente
Para usar: definir simbolo como global no outro programa

Relocação:
Carregador relocador ou carregador relativo
2 métodos:
1.Registro de modificação: descreve cada parte do codigo que deve ser trocada quando ocorre relocação ;
2.máscara de bits: um bit/yte de relocação é associado com cada palavra do objeto
A para absoluto – não precisa relocação
R para relativo – precisa relocação
LIGADOR CARREGADOR DE DOIS PASSOS

P1:
-aloca e distribui para cada programa endereços de memoria
- cria a tabela de simbolos, preenchendo os valores dos simbolos externos
P2:
- carrega o código do prog
- executa as modificações nos endereços (se necessarias)
- resolve referencias externas (de ligação)
TABELAS DE SIMB EXTERNOS:
-armazena nome e endereço dos simbolos externos referenciados
- indica o programa no qual o simbolo esta definido
ENDERECO DE CARGA PROGRAMADA
-Endereço inicial de memoria onde o programa ligado é pra ser carregado
- Fornecido pelo O/S
CONTROL SECTION ADDRESS
- Endereço inicial atribuido ao cs analisado pelo carregador
- seu valor é adicionado a todo endereço relativo, para converte-lo no atual
PASSO 2
- Faz a carga, relocação e ligação
- Cada registro de texto é lido e é transferido o controle para o programa carregado começar a execução