

Sistemas Operacionais

Trabalho Prático 1 – *Compact Threads* (*cthread*s)

Compilação, ligação e makefile
Primitivas *context*

Aula TP1

Biblioteca de threads *cthread*s 16.2

- Implementação de uma biblioteca de *threads* modelo N:1
 - Linguagem de programação C (não C++)
 - Executar em ambientes GNU/Linux (máquina virtual *alunovm-sisop.ova*)
 - Ferramenta VirtualBox (<https://www.virtualbox.org>)
- Principais tarefas
 - Desenvolvimento de um escalonador com política por loteria, primitivas para criação e sincronização de *threads* e programas de testes
 - Produto final: biblioteca *libcthread.a*
- Entregáveis e prazos
 - Arquivo *tar.gz* via Moodle (não pode *rar* ou similares) com o nome e conteúdo conforme a especificação
 - Data de entrega: 29 de setembro de 2016

Pode ser feito em duplas (de dois)!

Sistemas Operacionais

2

Primitivas *cthread*s

Primitivas para criação, liberação voluntária, sincronização de término (*join*), sincronização de threads (semáforos) e uma função genérica de identificação do grupo.

```
int ccreate (void *(*start)(void *), void *arg);
int cyield (void);
int cjoin (int tid);
int csem_init (csem_t *, int count);
int cwait (csem_t *sem);
int csignal (csem_t *sem);
int cidentify (char *name, int size);
```

Sistemas Operacionais

3

Exemplo de uso de *cthread*s

```
#include "../include/cthread.h"
#include <stdio.h>

void* func0(void *arg) {
    printf("Eu sou a thread ID0 imprimindo %d\n", *((int *)arg));
    return;
}

void* func1(void *arg) {
    printf("Eu sou a thread ID1 imprimindo %d\n", *((int *)arg));
}

int main(int argc, char *argv[]) {
    int id0, id1, i;

    id0 = ccreate(func0, (void *)&i);
    id1 = ccreate(func1, (void *)&i);

    printf("Eu sou a main após a criação de ID0 e ID1\n");

    cjoin(id0);
    cjoin(id1);

    printf("Eu sou a main voltando para terminar o programa\n");
}
```

Supõe que esteja no diretório testes

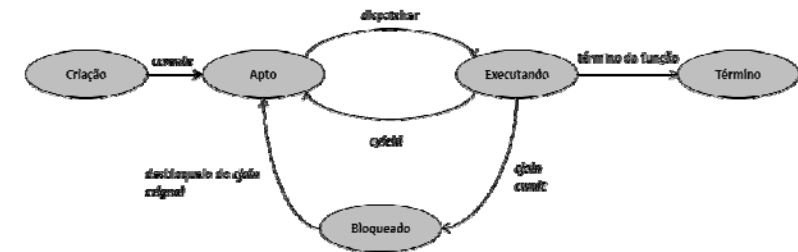
Sistemas Operacionais

4

Escalonador por loteria

- Escalonador não preemptivo
 - Perde o processador por chamada de sistema
 - Cedência voluntária, sincronização de término de *thread*, sincronização de *thread* ou término da função
- Emprega uma política de sorteio (escalonador lotérico)
 - Thread* recebe um bilhete de loteria na sua criação (valor de 0 a 255 aleatório)
 - Quando acionado, o escalonador gera um número aleatório (0 a 255)
 - A *thread* no estado apto que tiver o número mais próximo do número sorteado ganha o direito de executar na CPU
 - Em caso de empate, a que tem menor identificador (*tid*), ganha!

Diagrama de estados e transições de *threads*



Fila* aptos (uma só)
Fila* de bloqueados (uma por semáforo)
Fila de executando

*Na verdade são listas encadeadas!!

support.h
support.o

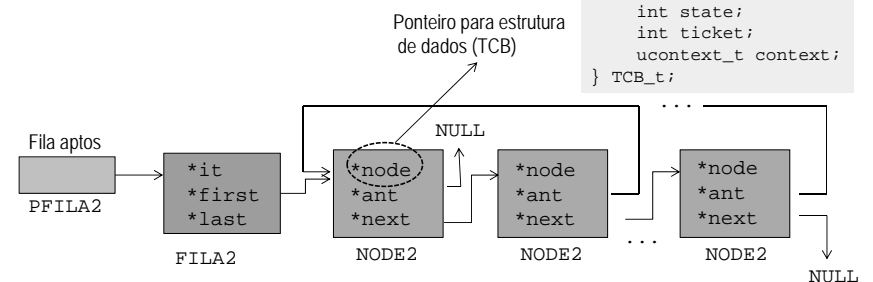
```
typedef struct s_TCB {
    int tid;
    int state;
    int ticket;
    ucontext_t context;
} TCB_t;
```

cdata.h

Material de apoio

- Biblioteca de funções *support*
 - Funções genéricas para tratamento de filas (criação, inserção, retirada, etc)
 - Função para geração de número pseudo-aleatório de 8 bits (0 a 255)
- Disponibilizada na forma de:
 - Arquivo binário *support.o* com o corpo das funções de fila e randomização
 - Deve ser ligado ao código desenvolvido
 - Arquivo de inclusão *support.h*
 - Prototypes* das funções de fila e randômica e suas estruturas de dados
 - Usado em arquivos fontes com suas funções ou estruturas de dados
- Arquivos de inclusão
 - thread.h*: NÃO pode ser modificado (*prototypes* das funções para usuário)
 - cdata.h*: estrutura de dados TCB e as serem criadas por vocês

Filas da *support.o*



```
struct sFila2{
    struct sFilaNode2 *it;
    struct sFilaNode2 *first;
    struct sFilaNode2 *last;
} Fila2
```

```
PNode2 = struct sFilaNode2
PFila2 = struct sFila2
```

```
struct sFilaNode2{
    void *node;
    struct sFilaNode2 *ant;
    struct sFilaNode2 *next;
} Node2
```

(Fornecidos no support.h)

Revisão de programação em C

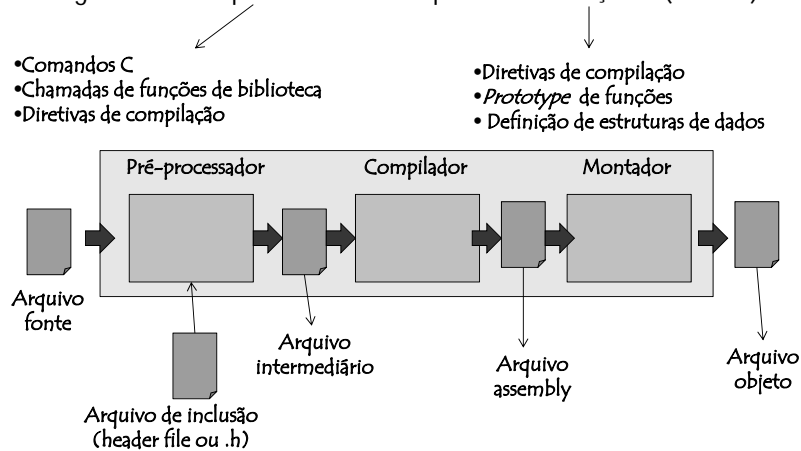
- Programas fontes C
 - Compostos por comandos C, tipos de dados, chamadas de função
 - Diretivas de compilação: ordens de como compilar um programa
 - Inclusão, definição de constantes, condicional
 - Iniciadas pelo caracter #
- Arquivos de cabeçalhos (*header files*)
 - Definição de *prototypes* de função (nome, tipo de retorno, parâmetros, etc...)
 - Definição de tipos de dados
 - Adicionados no momento da compilação através da diretiva *include*

Arquivos objetos e executáveis

- A compilação correta de um arquivo fonte gera um arquivo objeto
- Arquivos objeto
 - NÃO podem ser executados diretamente é necessário gerar um executável
 - Vários objetos podem ser combinados para gerar um único executável
 - Situação comum: objeto do programa fonte e biblioteca
 - Ex.: programa "Hello World!" possui um objeto e só se torna executável após ser combinado com a *standard library* (*stdlib* ou *stdio*)
 - hello.o* tem 1028 bytes e *hello* (executável) tem 7159 bytes (máquina virtual)
- A ligação é responsável por gerar um arquivo executável
 - Etapas obrigatória
 - Comportamento *default* dos compiladores (e.g. *gcc*)
 - Compilar e ligar já gerando um executável

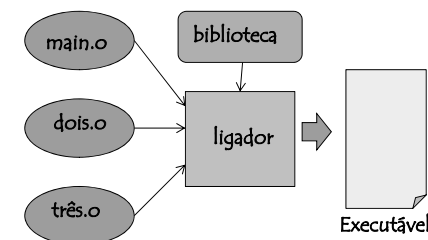
Desenvolvimento de programas em C (em Unix)

- Programa C = arquivos fontes + arquivos de cabeçalho (header)

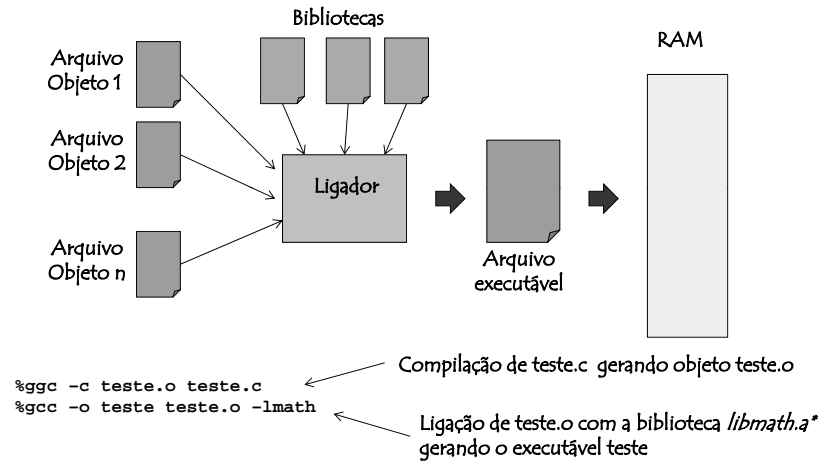


Ligador

- Combinação de vários arquivos objetos para gerar um executável
 - Arquivos desenvolvidos pelo próprio usuário ou por terceiros (bibliotecas)
 - Objetivo é resolver as referências externas



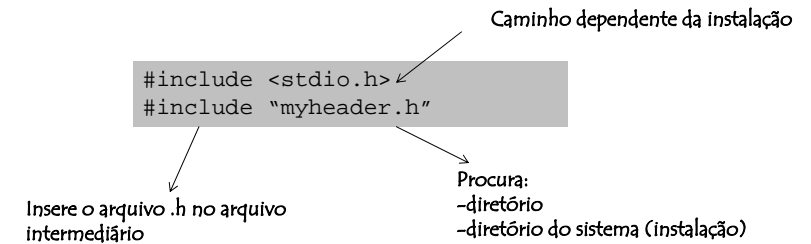
Desenvolvimento de programas em C (em Unix)



* A localização da biblioteca é dada por um caminho padrão e possuem nome que é prefixado por lib com a extensão .a (se estáticas)
Sistemas Operacionais

Diretivas de compilação

- Extensão fornecida pelo compilador, não pela linguagem
- Possuem um símbolo identificador (#)
- Sintaxe independente do C
- Podem ser inseridos em qualquer parte do programa fonte
- Não possuem regras de escopo



Sistemas Operacionais

14

Alguns outros exemplos de diretivas

```
#define CONST 5
#undef CONST
#define swap(a,b) ((a)>(b)?(a):(b))

#if MAX_PROC
    s=random(c);
#define MIN 10
#else
    s=random(2);
#endif

#ifndef PENTIUM
    [...comandos C...]
#endif

#if TAB_SIZE == 100
    int y=10;
#endif
```

Sistemas Operacionais

15

Makefile

- Mecanismo para criar programas executáveis e gerenciar arquivos
 - Uso comum: realizar a compilação condicional de arquivos fontes e gerar executáveis
 - Verifica de acordo com um critério de data de modificação e dependência quais ações devem ser executadas
- Duas partes
 - Comando *make* e similares (*gmake*, *pmake*, etc)
 - Arquivo de makefile (*makefile* ou *Makefile*)
 - Regras com as dependências

Sistemas Operacionais

16

Comando *make*

- Lê um arquivo de *makefile* e determina qual ou quais *targets* (arquivos) precisam ser gerados
 - Critério: comparação entre a data e hora de criação e alteração dos arquivos
 - Arquivos que dependem de arquivos com hora de modificação mais recente precisam ser regerados
- Princípio de funcionamento
 - O arquivo de *makefile* possui alvos (*targets*)
 - Inicia no alvo nomeado como *all*, se não houver alvo *all* inicia
 - no primeiro alvo do arquivo
 - no alvo fornecido como argumento do comando *make*

17

Arquivo de makefile

- Conjunto de dependências e regras
 - Basicamente, target (quem vai ser criado) e uma lista do conjunto de arquivos de quem ele depende (dependências)
- Formato geral:

```
Target: dependências  
[tab] comandos
```

SEM espaços em branco ao final!!

Atenção: é tab!! Não é espaço em branco!!

Sistemas Operacionais

18

Exemplo (simples) de makefile e sua execução

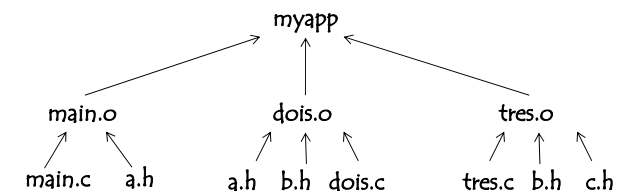
```
myapp: main.o dois.o tres.o  
    gcc -o myapp main.o dois.o tres.o  
  
main.o: main.c a.h  
    gcc -c main.c  
  
dois.o: a.h b.h dois.c  
    gcc -c dois.c  
  
tres.o: tres.c b.h c.h  
    gcc -c tres.c  
  
clean:  
    rm *.o
```

Exemplos:

make clean (executa a regra com nome clean)
make (executa a regra *myapp*, primeira do arquivo)
make dois.o (inicia pela regra *dois.o*)

19

Exemplo (simples): dependência



Situação exemplo:

Se o arquivo *b.h* for modificado, ele terá uma data mais recente que os demais. Assim, ao se executar *make* (ou *make myapp*), como *myapp* depende de *dois.o* e de *tres.o* e esses dois, por sua vez, dependem de *b.h*, será executado as regras *dois.o* e *tres.o*. Note que *main.o* não será executado por depender de *b.h*.

Sistemas Operacionais

20

Exemplo de makefile: alguns detalhes

```
#comentários iniciam o caracter sustentado
#Possível definir macros (ex.: nome_da_macro = valor)

CC = gcc
LIBS = -L. -l/user

all: myapp # por onde iniciar a análise de dependência
          # se não há all, inicia pela primeira regra

myapp:    # pode não haver dependência
          $(CC) -o myapp myapp.c $(LIBS)/libl
```

Família de funções *ucontext*

- Chamadas de sistema do Linux
 - Manipulação de contexto em modo usuário
 - Permitem a criação e chaveamento de contextos
- Funções existentes
 - *makecontext()*: criação
 - *getcontext()*: salva contexto de execução
 - *setcontext()*: recupera contexto de execução
 - *swapcontext()*: troca contexto de execução
 - Equivalente a fazer um *getcontext* seguido de um *setcontext*
- Estrutura de dados *ucontext*
 - Define uma pilha, função associada ao contexto, informações do contexto

getcontext()

```
getcontext(ucontext_t *ucp);
```

- Inicializa a estrutura apontada por *ucp* para o contexto do fluxo de execução que a executa
 - Registradores (PC, SP e GPRs), máscara de sinais e pilha
- O contexto *ucp* pode ser usado para:
 - Opção I: salvar contexto que pode ser retornar posteriormente via *setcontext()*
 - Opção II: servir para criar um molde de contexto, o qual será posteriormente modificado por uma chamada de *makecontext()*
 - Análogo a criar um processo filho como cópia do pai e, posteriormente, trocar seu código.

makecontext()

```
makecontext(ucontext_t *ucp, void (*func)(), int arg,...);
```

- Modifica o contexto especificado por *ucp* o qual foi inicializado por uma chamada prévia a *getcontext()*
- Antes de chamar *makecontext* é preciso modificar *ucp* para
 - Definir uma pilha (área e tamanho)
 - Inicializar o campo *uc_link* para indicar o contexto (*ucp*) a ser executado quando o contexto definido por *makecontext* terminar sua execução.
- *makecontext* define
 - Uma função a ser executada pelo contexto (*func*)
 - Quantidade de argumentos (*argc*) e a lista de argumentos (todos inteiros) a serem passado para o contexto.

setcontext()

```
setcontext(ucontext_t *ucp);
```

- Restaura o contexto apontado por *ucp*
 - Retorna a um ponto (contexto) previamente definido
 - Contexto deve ter sido criado previamente por *getcontext()* ou *makecontext()*
- O contexto *ucp* foi definido por:
 - getcontext()*: execução continua na instrução seguinte após a chamada do local onde *getcontext* foi feito.
 - makecontext()*: execução continua na função definida pelo primeiro argumento de *makecontext*.

swapcontext()

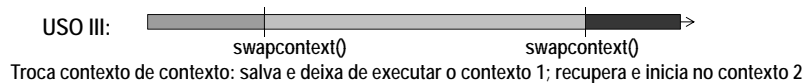
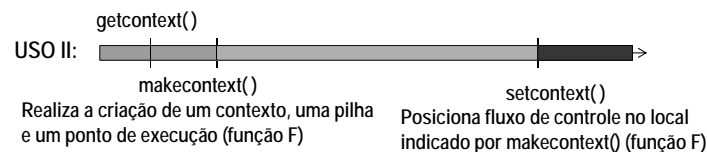
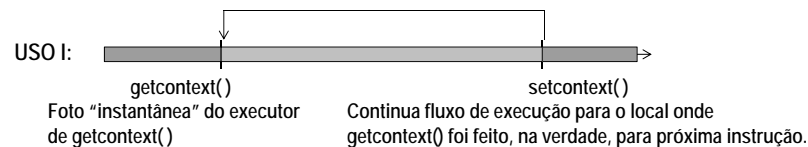
```
swapcontext(ucontext_t *oucp, ucontext_t ucp);
```

- Salva o atual contexto de execução em *oucp* e desvia para o contexto apontado por *ucp*
- Efetua a troca de dois contextos

Chamadas de sistema Linux para tratamento de contexto

FUNDAMENTAL para o trabalho prático da disciplina

- Funções *makecontext()*, *setcontext()*, *getcontext()* e *swapcontext()*



Referências iniciais

- Roteiro experimental da atividade no moodle com exemplos
- Man page das funções
- <http://en.wikipedia.org/wiki/makecontext>