UNIVERSIDADE ESTADUAL DO RIO GRANDE DO SUL

UNIDADE GUAÍBA

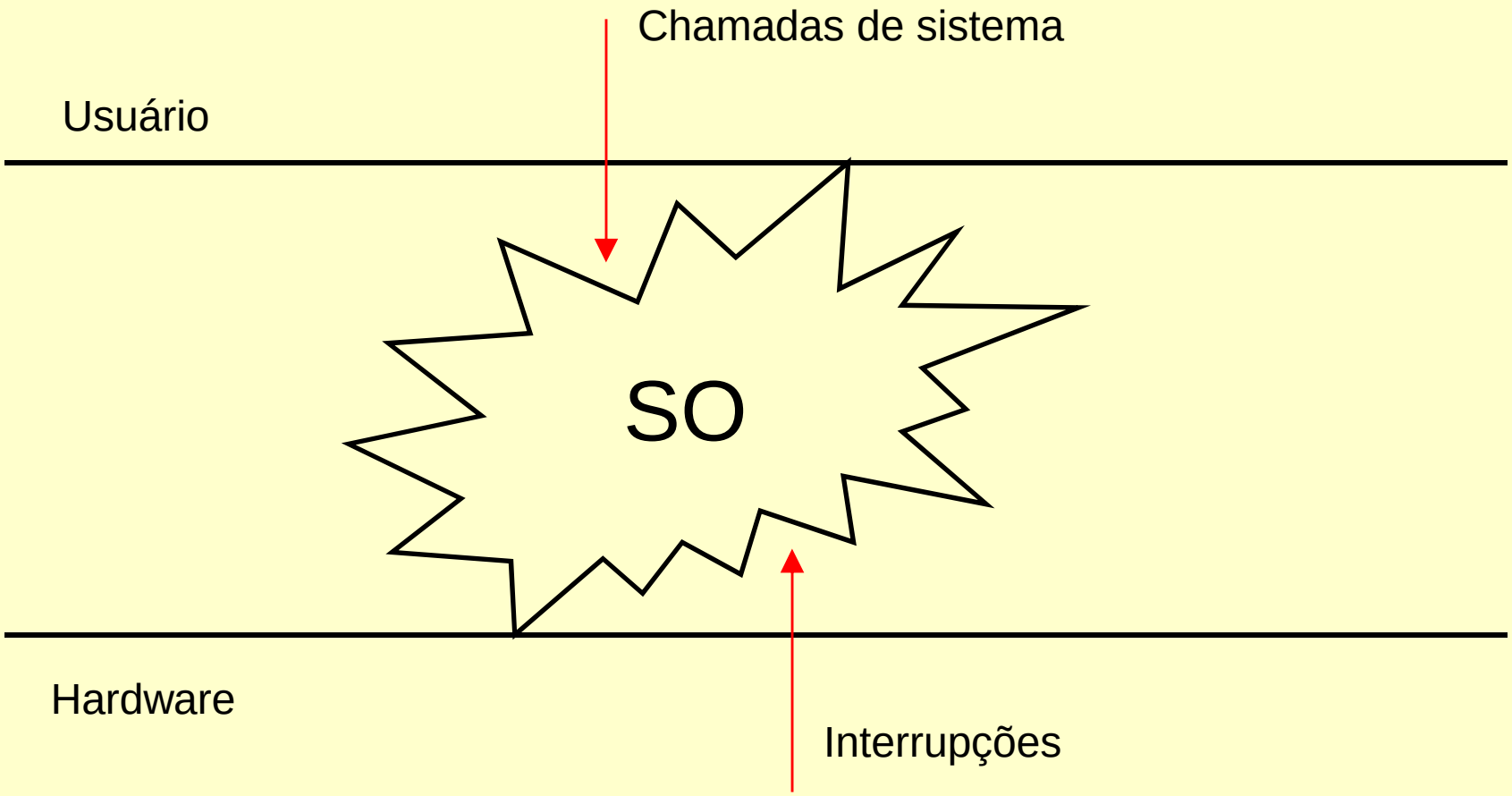
Programação de Sistemas

Chamadas de Sistema

Celso Maciel da Costa

Guaíba, Novembro 2014.

Visão Esquemática do funcionamento de um SO



Chamadas de sistema

- interface entre o programa que está executando e o sistema operacional
  - **funções do SO disponíveis aos programas**
- geralmente implementadas com o uso de instruções de baixo nível

Chamadas de sistema

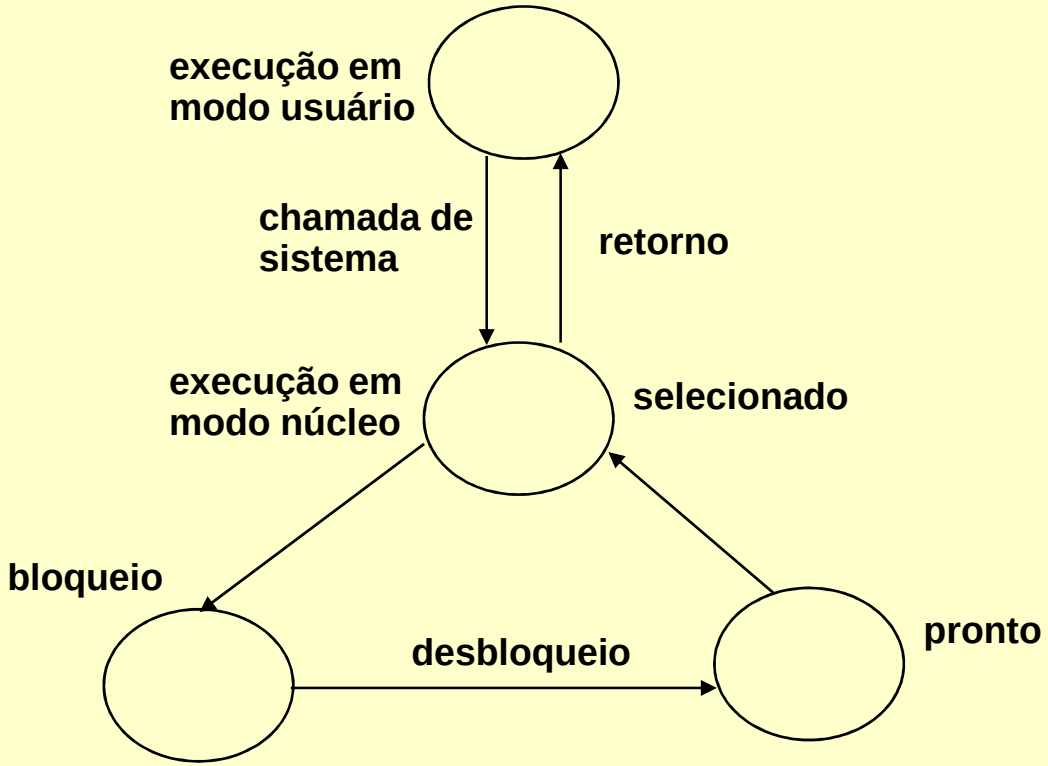
- Controle de processos
  - **criar, terminar**
  - **carregar, executar**
  - **esperar por evento, sinalização**
  - **esperar tempo**
  - **alocar e liberar memória**

Chamadas de sistema

- Manipulação de arquivos
  - **criar, deletar**
  - **abrir, fechar**
  - **ler, escrever**
  - **posicionar**

Chamadas de sistema

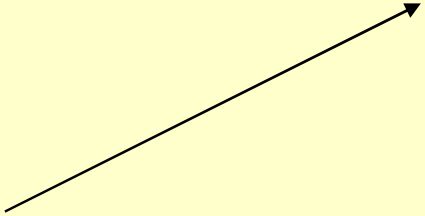
- Manipulação de dispositivos
  - **alocar dispositivos, liberar dispositivo**
  - **ler, escrever**
- Manutenção de informação do sistema
  - **e.g.: ler, setar a hora**
- Comunicação
  - **criar, deletar canais de conexão**
  - **transferir informação**



Transição de estados de um processo em uma chamada de sistema

Programa

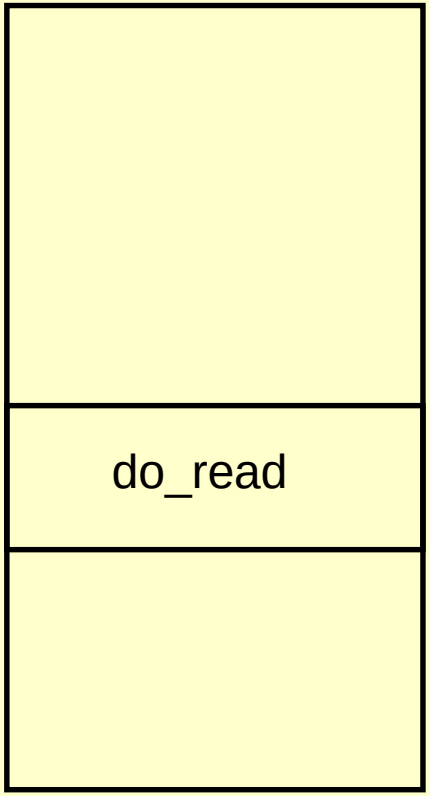
```
...
...
...
read ( fd, $b, 20)
```



Biblioteca

```
read ( int fd, char *b, int l )
{
    msg  *m;
    m->fd = fd;
    m->b = b;
    m->l = l;
    m->op = READ;
     move m, regA;
    trap #12;
}
```

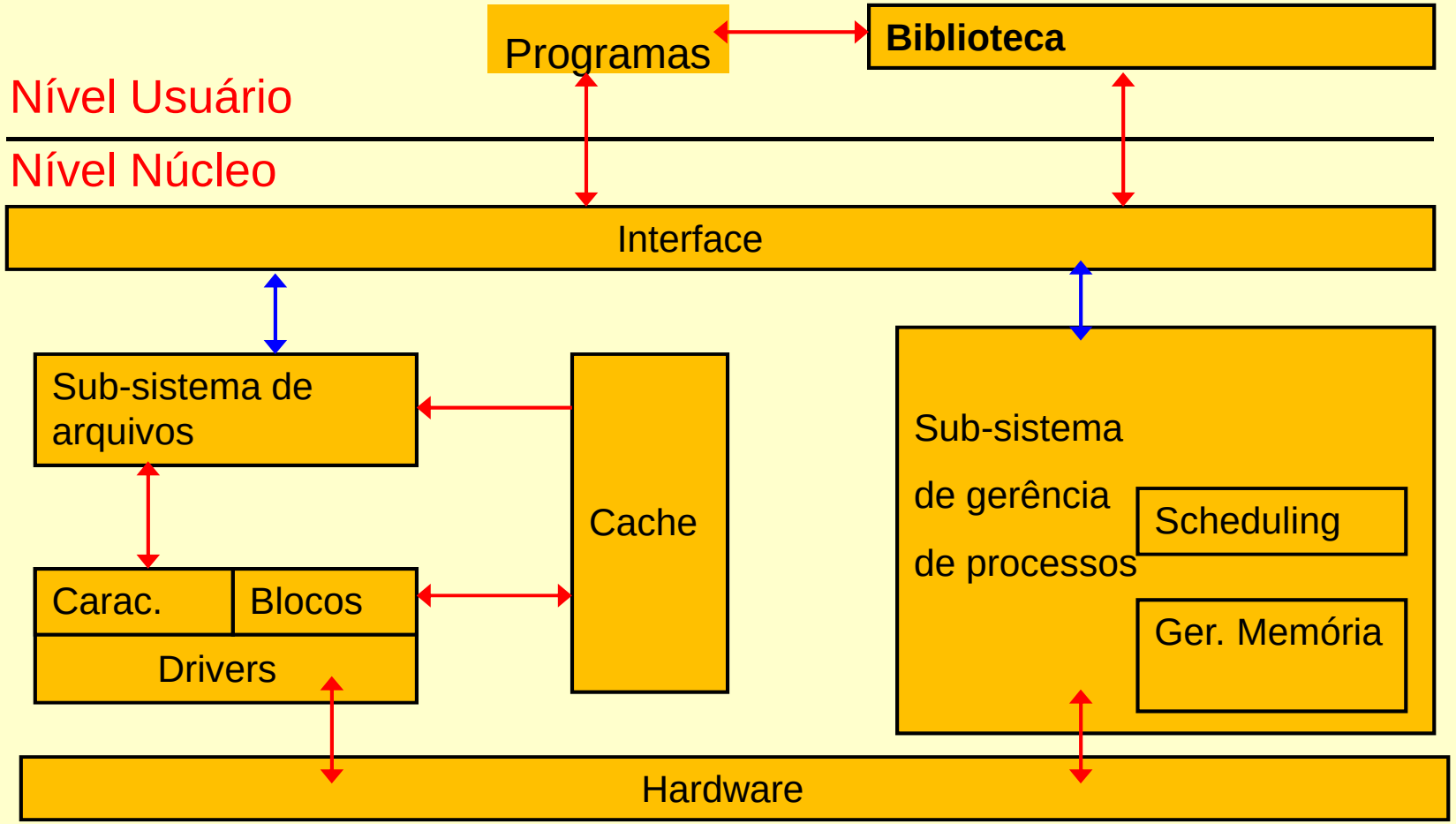
Vetor interrupções




```
do_read: desabilitar int
         salvar contexto
         “executar função”
          ...
         return ;
```

Chamadas de sistema - Unix

- divididas em dois grandes grupos:
  - **controle de processos, manipulação de arquivos**



Chamadas de sistema - Unix

Manipulação de arquivos

- . open
- . close
- . read
- . write
- . create
- . pipe
- . link
- . unlink
- . lseek

Chamadas de sistema - Unix

Controle de processos

. fork

. wait

. exit

. exec

. kill

. signal

Chamadas de sistema

- Implementação
  - **Depende do computador**
  - **Exemplo:**
    - » **instrução especial transfere o controle para o SO**
    - » **chamada de uma rotina especial para executar o serviço**

NKE

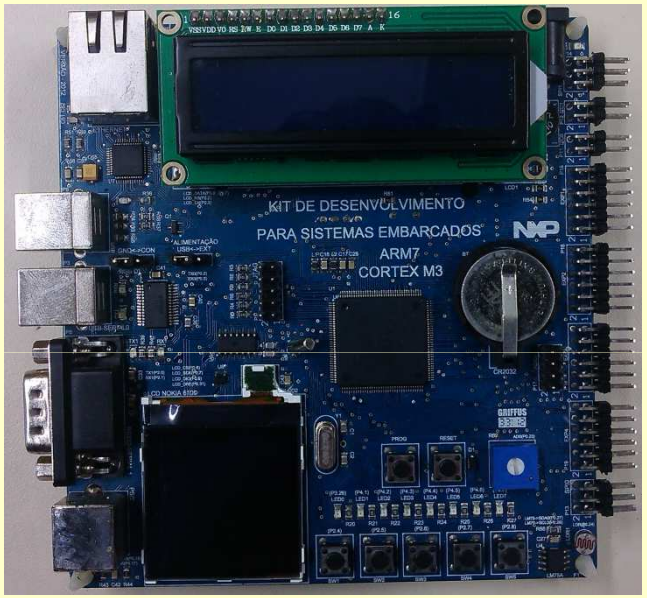
- Sistema operacional voltado para o ensino, e a aprendizagem com o objetivo de permitir o desenvolvimento de aplicações para sistemas embarcados e de tempo real.

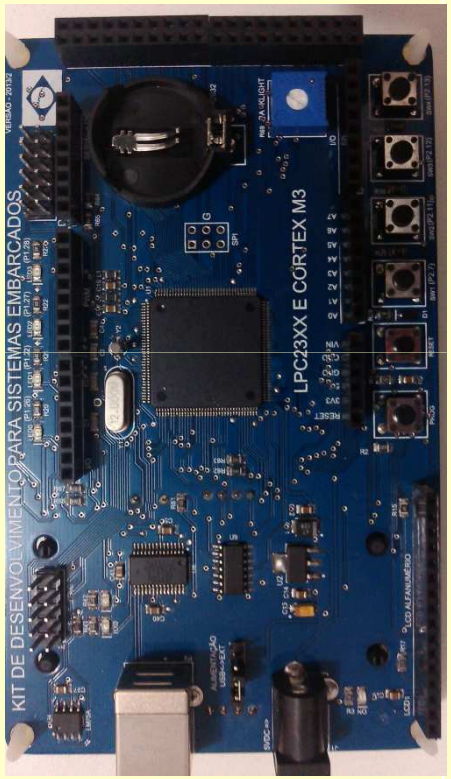
NKE - Nanokernel

- Um Kernel geralmente é um componente básico do sistema operacional, oferecendo uma camada de abstração de nível mais baixa para os recursos (especialmente processadores e dispositivos de entrada/saída).

- Um Nanokernel é uma camada de abstração de hardware que constitui a parte mais baixa do nível de um Kernel.

Ambiente de desenvolvimento





LPC2378

- Microprocessador ARM7TDMI
- 512K Bytes de Memória Flash programável
- 56K Bytes de RAM
- Quatro UARTs
- Quatro temporizadores
- Relógio de tempo real, RTC

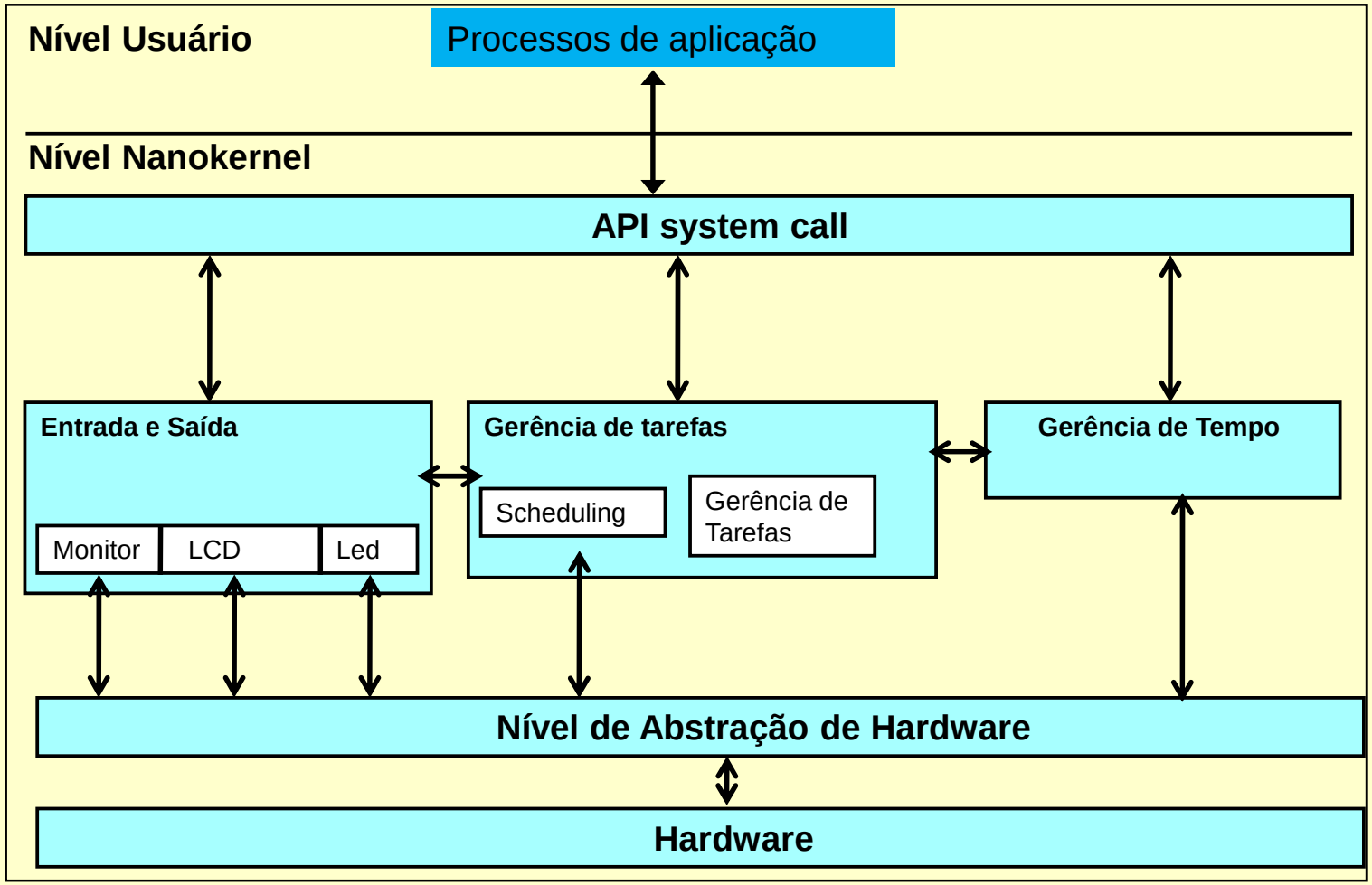
Raspberry

BeagleBone

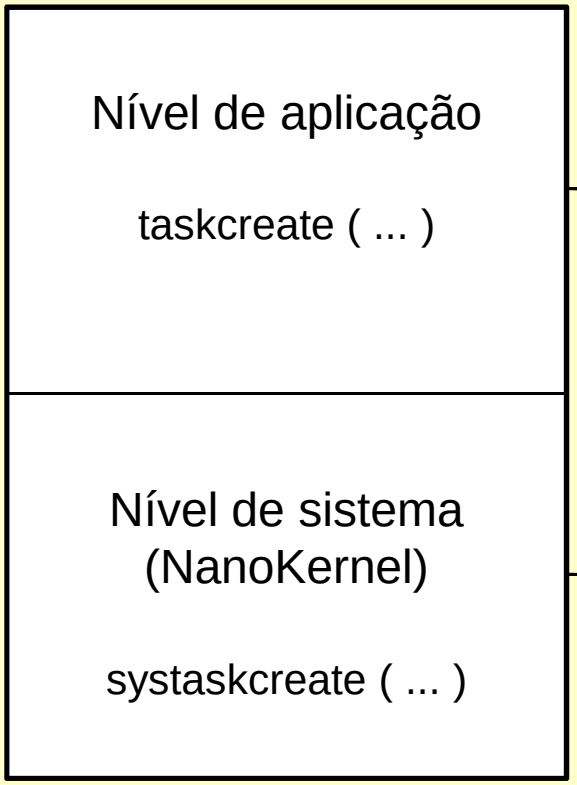
Arquitetura do NKE

- Uso do NKE: o programa é escrito em C e ligado ao *Nanokernel* antes de ser carregado para execução;
- O NKE trabalha com dois níveis lógicos:
  - ***kernel* e *user -level***
- O processo (em *user-level*) chama os serviços do NKE (em *kernel-level)* por meio de *system calls*;
- O acionamento de *system calls* é implementado usando interrupções de software, via operação SWI do ARM;
  - ***System call stubs* encapsulam esse mecanismo, deixando-o transparente para o programador.**

Arquitetura do NKE



Organização do Sistema



Utiliza o modo usuário do ARM.

É o nível lógico do programa aplicação. Neste nível, uma aplicação só irá realizar o acesso aos recursos do sistema através de requisições.

Utiliza o modo supervisor do ARM.

É o nível que virtualizará os recursos de hardware de maneira simples e mínima. Utilizará tais módulos para disponibilizar recursos para os programas aplicação.

Nível Aplicação – Tarefa

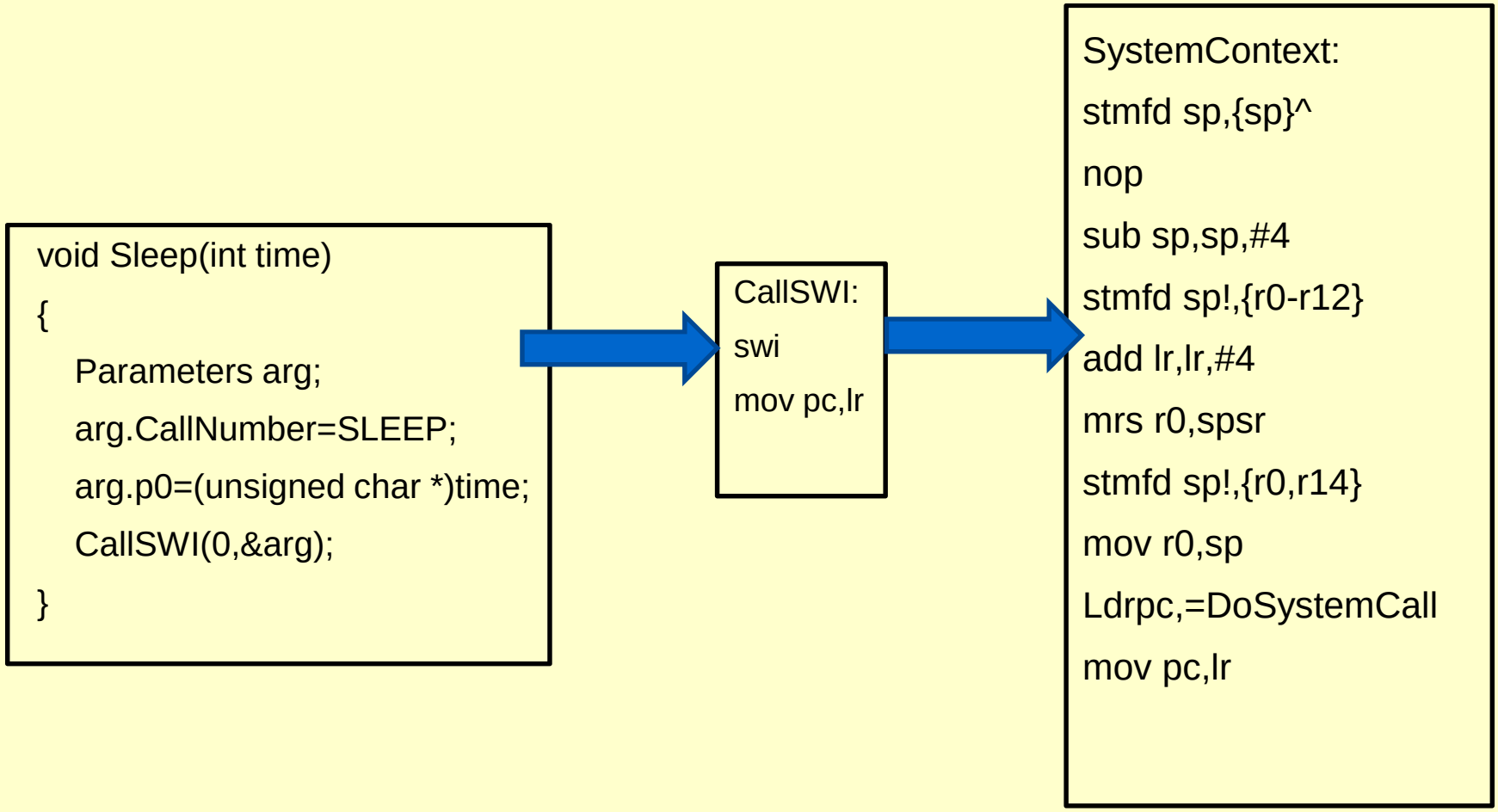
- Tarefa (*task):*
  - **É definida como sendo a execução de um fluxo seqüencial de instruções, construído para atender uma finalidade específica.**

Exemplo de aplicação

```
#include “../Kernel/kernel.h”

int t1,t2,soma=0;
sem_t s0;

void soma_1()
{
    int i;
    for( i=0; i<1000; i++)
    {
            soma = soma + 1 ;
            Sleep(2);
    }
    ExitTask();
}
```

```
void soma_2()
{
    int i;
    for( i=0; i<1000; i++)
    {
            soma = soma + 2 ;
    }
    ExitTask();
}

int main(int argc, char *argv[])
{   taskcreate(&t1,soma_1);
    taskcreate(&t2,soma_2);
    start(RR);
}
```

Chamadas de Sistema



Chamadas de Sistema

```
void DoSystemCall(unsigned int *stack, Parameters *arg)
{
   Descriptors[TaskRunning].SP=stack;
   MoveToSP(&KernelStack[289]);
   switch(arg->CallNumber)
   {
     ...
     case EXITTASK:
        sys_TaskExit();
        break;
     case SLEEP:  // <---------- Case do Sleep
        sys_Sleep((int)arg->p0);
        break;
     ...
   }
  RestoreContext(Descriptors[TaskRunning].SP);
}
```

```
void sys_sleep(unsigned int segundo)
{
   Descriptors[TaskRunning].Time = segundo/ClkT;
   if(Descriptors[TaskRunning].Time > 0)
   {
       Descriptors[TaskRunning].State = BLOCKED;
       Dispatcher();
    }
}
```

Entrada e Saída de Dados

- É a saída de dados do NKE.

- Têm quatro chamadas de sistema:
  - **writeLCDN(int *number, int pos);**
  - **writeLCDS(char *string, int pos);**
  - **LEDON(int value);**
  - **nkprint (char *string, void *value);**
  - **nkread(char* type, void *value)**

Gerência de Tarefas

- Corresponde a sincronização, escalonamento, criação e termino das tarefas.

- Possui seis chamadas de sistema:
  - **seminit (sem_t &semaphore, int value);**
  - **sempost (sem_t &semaphore);**
  - **semwait (sem_t &semaphore);**
  - **taskcreate (int &Tid, void &(TaskEP)());**
  - **taskexit ( )**
  - **start (int Scheduler)**

Gerência de Tempo

- Possui três chamadas de sistema:
  - **sleep (int value);**
  - **msleep (int value);**
  - **usleep (int value);**