TCD – Trabalho de Conclusão de Disciplina

GBC045 – Sistemas Operacionais

Grupo 04:

Antonio Carlos Neto ---------- 11611BCC054

Higor Emanuel Souza Silva ---------- 11611BCC016

Marcelo Mendonça Borges ---------- 11611BCC020

Problema Proposto

A System Call deve receber como parâmetro o valor de um PID e retornar:

-> O número de processos filhos desse processo identificado pelo PID;

-> Para cada processo filho retornar:

- O PID do processo.
- O PPID do processo.
- O UID do processo.

-> (-1) se o processo (PID) não existir.

A System Call

```
asmlinkage long sys_tcd_g4(int __user pid, int __user *buf, int __user size);
```

Parâmetro “pid”: valor do PID lido no programa usuário.

Parâmetro “buf”: vetor(buffer) criado no programa usuário que recebe os valores de retorno da syscall.

Parâmetro “size”: tamanho em bytes do vetor(buffer).

Acessando o PCB de um Processo

```
struct pid *pid_struct;
struct task_struct *task;
```

```
pid_struct = find_get_pid(pid);
```

Obtém a *struct pid* referente ao PID passado de parâmetro.

```
task = pid_task(pid_struct,PIDTYPE_PID);
```

Obtém a *struct task_struct* referente a *struct pid* passada de parâmetro.

Acessando os Filhos do Processo Principal

```
#define list_for_each(pos, head) \
        for (pos = (head)->next; pos != (head); pos = pos->next)
```

```
struct list_head *i;
```

```
list_for_each(i, &(task->children))
```

Percorre uma lista bilateral circular de *struct list_head*.

Acessando os Filhos do Processo Principal

```
struct task_struct *ts;
```

```
ts = list_entry(i, struct task_struct, sibling);
```

Realiza a conversão de uma *struct list_head* para a estrutura que o contém, que, no caso, é a *struct task_struct*.

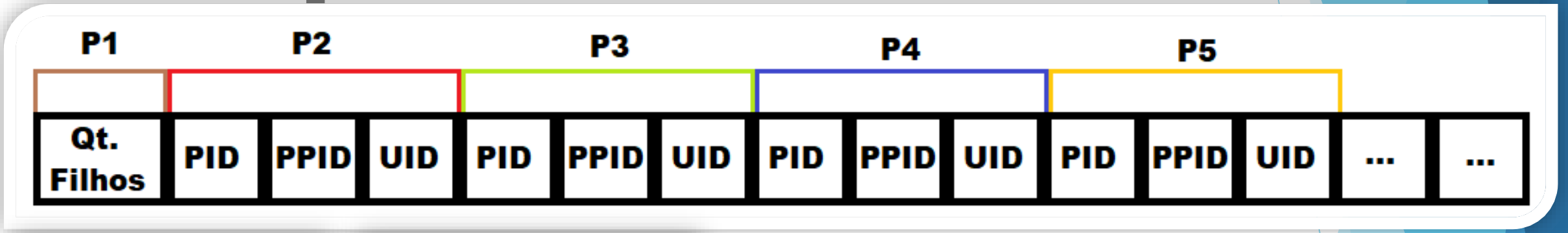
Obtendo os valores do PID, PPID e UID dos Processos

```
struct task_struct *ts;
```

```
kpid = ts->pid;
kppid = ts->parent->pid;
kuid = ts->cred->uid.val;
```

O PID é obtido diretamente na *struct task_struct*, o PPID é obtido da *struct task_struct* do pai, e o UID é obtido na *struct cred* que é referenciada dentro da *struct task_struct*.

Passando os Dados para o User Space




```
int aux;
```

```
int deep;
```

Variável “aux”: Contabiliza a quantidade de filhos do processo.

Variável “deep”: Contabiliza a quantidade de filhos cujos dados não são escritos no vetor(buffer).

Passando os Dados para o User Space

```
copy_to_user(&buf[((aux*3)+1)], &kpid, sizeof(int))
```

```
copy_to_user(&buf[((aux*3)+2)], &kppid, sizeof(int))
```

```
copy_to_user(&buf[((aux*3)+3)], &kuid, sizeof(int))
```

```
copy_to_user(&buf[0], &aux, sizeof(int))
```

Função “copy_to_user”: Copia o conteúdo de um endereço de memória para outro, ou seja, transferindo informações do Kernel Space para o User Space.

Programas Auxiliares

-> p1.c

Criamos um programa que executa um processo que cria 5 filhos, e fica rodando na memória.

-> t.c e t1.c

Criamos dois programas usuário que alocam o vetor(buffer) e passa de parâmetro para a syscall. A diferença entre eles é o tamanho do buffer.