PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

PUC Minas

Sistemas operacionais

Curso: Engenharia da Computação

Autores: Isabela Ferreira Scarabelli, Lucas José de Freitas e Pedro Henrique de Almeida Santos

Documentação “Implementando uma Chamada de Sistema no Nanvix”

Objetivo:

- Visando a adição de uma nova *feature* para o Sistema Operacional Nanvix, foi desenvolvido pelos alunos a função ***getp***, pensada como a implementação de uma chamada de sistema que tenta recuperar informações de um determinado processo, sendo elas o estado atual do processo, o seu nível de prioridade, o seu tempo de CPU em modo usuário e em modo Kernel, buscada pelo ***pid*** do processo e as armazenada em ***process_buf***, que desta forma, se encontrado retorna todas essas informações ao usuário, caso contrário retorna -1. Para que assim, o usuário tenha controle e ciência de todos os processos que estão rodando em sua máquina.

Implementação:

- A estrutura de dados ‘***process_buf***’ baseada na própria estrutura do Nanvix ‘***process***’. A struct ‘***process_buf***’ contém: o ***identificador*** do processo (do tipo pid_t), sendo ele o **pid**, o estado atual do processo (do tipo *signed*), ***state***, a prioridade (tipo *int*), ***priority***, o tempo de CPU do usuário (do tipo *unsigned*)**, *utime*,** e o tempo de kernel (do tipo *unsigned*), ***ktime***. Essa *struct* foi implementada no arquivo ***include/nanvix/pm.h***. Explicitando também os estados possíveis de um processo, sendo eles, ***DEAD*** mostra quando a execução do processo terminou, ***ZOMBIE*** indica que o processo terminou, mas ainda não teve sua remoção completa, ***RUNNING*** indica quando o processo está em execução, ***READY*** indica quando o processo está pronto para ser executado**, *WAITING*** indica que um processo está aguardando algum evento para continuar sua execução, ***SLEEPING*** indica que o processo em estado de espera aguardando algum evento e ***STOPPED*** indica que um processo foi interrompido por algum motivo.

```
/**
 * @brief Process buffer. Utilizado para a instrução get_process_info()
 */
struct process_buf{
        pid_t pid; // process ID
        unsigned state; //current state
        int priority; //process priority
        unsigned utime; //user CPU time
        unsigned ktime; //kernel CPU time
};

/* States:
DEAD (0)
ZOMBIE (1)
RUNNING (2)
READY (3)
WAITING (4)
SLEEPING (5)
STOPPED (6)
*/
```

- Implementação da chamada de kernel **sys_get_process_info()**, dentro arquivo **src/kernel/pm/get_proc_info.c**, parametrizada por **pid** e buf, que chama a função do_get_process_info(). Essa função é o primeiro acesso com o kernel.

```c
#include <nanvix/klib.h>
#include <nanvix/pm.h>

PUBLIC void  sys_get_process_info( pid_t pid, struct process_buf *buf ){
    do_get_process_info( pid, buf );
}
```

- do_get_process_info: a função procura na tabela de processos (proctab) o processo que se refere ao pid requerido. Ao encontrá-lo, aponta buffer para o mesmo endereço e imprime as informações na tela para o usuário. Conforme já definido pelo Nanvix, FIRST_PROC representa o processo da primeira posição de proctab, e LAST_PROC representa o último processo. A variável 'aux' serve apenas como auxiliar para determinar em qual posição da tabela de processos está o processo necessário. Ela foi implementada no arquivo src/kernel/pm/pm.c.

```c
/*
Salva as informações da struct process do pid requerido,
em buf, e depois mostra na tela o resultado para o usuário.
*/
//pid_t --> typedef de signed
PUBLIC void do_get_process_info(pid_t pid, struct process_buf *buf){
        struct process *proc;
        int aux = 1; //salva a posição de proctable que aquele processo ocupa

        //Loop na tabela de processos
        for (proc = FIRST_PROC; proc <= LAST_PROC; proc++){
                //Checa se aquele processo possui o pid requerido
                if(proc->pid == pid){
                        buf->pid = proctab[aux].pid;
                        buf->state = proctab[aux].state;
                        buf->priority = proctab[aux].priority;
                        buf->utime = proctab[aux].utime;
                        buf->ktime = proctab[aux].ktime;
                }

                aux++;
        }

        char p[26];
        char priority[26];
        char utime   [26];
        char ktime   [26];

        const char *states[7];
        states[0] = "DEAD";
        states[1] = "ZOMBIE";
        states[2] = "RUNNING";
        states[3] = "READY";
        states[4] = "WAITING";
        states[5] = "SLEEPING";
        states[6] = "STOPPED";
```

As funções rvr, ips e pValue servem apenas para formatação do conteúdo da tela. Ele transforma os inteiros em caracteres e estão implementados juntos com do_get_process_info.

```
void rvr(char* s)
{
        int i, j;
        char c;

        for (i = 0, j = kstrlen(s)-1; i<j; i++, j--)
        {
                c = s[i];
                s[i] = s[j];
                s[j] = c;
        }
}
```

```
void ips(int n, char* s)
{
        int i, sign;

        if ((sign = n) < 0)
                n = -n;

        i = 0;
        do
        {
                s[i++] = n % 10 + '0';
        } while ((n /= 10) > 0);

        if (sign < 0)
                s[i++] = '-';

        s[i] = '\0';
        rvr(s);
}
```

```
void pValue(int value, char* s, int padding)
{
        int i,len,size;

        ips(value, s);
        len = padding - kstrlen(s);

        size = kstrlen(s);
        for(i=size; i<len+size; i++)
                *(s+i) = ' ';

        *(s+i) = '\0';
}
```

- Implementação do arquivo **getp.c**, cuja a função ***main()*** chama a função ***getargs*** para transformar o argumento passado em um do tipo ***int*** , em seguida chama ***getp()*** recebendo o ***pid*** do processo e a ***struct do buffer***, caso execução termine bem sucedida a função retorna 0, caso contrário retorna -1, para isso foi criada uma pasta em **src/ubin** e o arquivo **getp.c** em **src/ubin/getp**

```c
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <nanvix/pm.h>

/* Software versioning. */
#define VERSION_MAJOR 1 /* Major version. */
#define VERSION_MINOR 0 /* Minor version. */

/* Default Process name. */
#define PROCESS_DEFAULT 0

/*
 * Program arguments.
 */
static struct{
        struct process_buf *buf; //buffer with the informations of the process
        pid_t pid; /* ID of process. */
} args = { 0, 0 };

/*
 * Prints program version and exits.
 */
static void version(void)
{
        printf("getp (Nanvix Coreutils) %d.%d\n\n", VERSION_MAJOR, VERSION_MINOR);
        printf("Copyright(C) 2011-2014 Pedro H. Penna\n");
        printf("This is free software under the ");
        printf("GNU General Public License Version 3.\n");
        printf("There is NO WARRANTY, to the extent permitted by law.\n\n");

        exit(EXIT_SUCCESS);
}
```

```c
/*
 * Prints program usage and exits.
 */
static void usage(void)
{
        printf("Usage: getp [options] <pid>\n\n");
        printf("Brief: Sends a signal to a process.\n\n");
        printf("Options:\n");
        printf("  --help             Display this information and exit\n");
        printf("  --version          Display program version and exit\n");

        exit(EXIT_SUCCESS);
}

/*
 * Gets number of the process.
 */
static void getargs(int argc, char *const argv[])
{
        int i;          /* Loop index.          */
        char *arg;      /* Current argument.    */

        /* Read command line arguments. */
        for (i = 1; i < argc; i++)
        {
                arg = argv[i];

                /* Parse command line argument. */
                if (!strcmp(arg, "--help")) {
                        usage();
                }
                else if (!strcmp(arg, "--version")) {
                        version();
                }
                else {
                        args.pid = atoi(arg);
                }
        }
```

```c
int main(int argc, char *const argv[])
{
        getargs(argc, argv);

        if (getp(args.pid, &args.buf) > 0){
        return 0;
    } else {
        return -1;
    }
}
```

- Adição da função nos programas de usuário e incluída em **MakeFile**, para adicionar no sistema de **build** do Nanvix a nova função **getp**

```
.PHONY: getp
```

```
# Builds everything.
all: cat chgrp chmod chown cp echo kill ln login ls mv nice pwd rm stat sync \
        touch tsh ps clear nim sleep getp
```

```
# Builds getp.
getp:
        $(CC) $(CFLAGS) $(LDFLAGS) -D TESTE getp/*.c -o $(UBINDIR)/getp $(LIBDIR)/libc.a
```

```
@rm -f $(UBINDIR)/getp
```

- Assinaturas das funções e seus respectivos arquivos onde foram implementadas:

- **do_get_process_info() - include/nanvix/pm.h**

```
/* Assinatura da função de kernel do_get_process_info() */
EXTERN void do_get_process_info(pid_t pid, struct process_buf *buf);
```

- **sys_get_process_info() - include/nanvix/syscall.h**

```
/*
 * Chamada de kernel que retorna para um buffer de usuário as informações de um processo.
 */
EXTERN void sys_get_process_info( pid_t pid, struct process_buf *buf );
```