SYSTEMY OPERACYJNE

Lab. – Minix wywołanie systemowe

Max Children

Wiktor Daszczuk wiktor.dasczuk@pw.edu.pl

Waldemar Grabski waldemar.grabski@pw.edu.pl

1 OPIS

Celem ćwiczenia jest dodanie w jądrze systemu Minix dwóch wywołań systemowych umożliwiających znalezienie procesu posiadającego największa liczbę dzieci i pobranie jego identyfikatora oraz liczby jego dzieci.

2 INSTALACJA ŚRODOWISKA

System operacyjny *Minix* w wersji 2.0.3 będzie uruchamiany w maszynie wirtualnej z wykorzystaniem *Qemu*.

Przykładowe środowisko dla systemu *Windows 11*.

Maszyna wirtualna *Hyper-V* z zainstalowanym systemem *Ubuntu* w wersji serwerowej (bez interfejsu graficznego).

W konfiguracji *Hyper-V*, tworzymy w wirtualnym przełączniku sieć wewnętrzną o nazwie *Internal*.

W konfiguracji maszyny:

- pierwszą kartę sieciową podłączamy do domyślnego przełącznika wirtualnego,
- dodajemy drugą kartę sieciową i podłączamy ją do sieci *Internal*.

Należy zainstalować wersję minimalną z zainstalowanym serwerem *SSH* (*openSSH*) co należy wybrać przy instalacji *Ubuntu*.

W systemie *Ubuntu*, karty sieciowe należy skonfigurować:

- *eth0 – DHCP*,
- *eth1* – statyczny adres IP 10.100.0.10/24 (nie określamy bramy).

Plik konfiguracyjny sieci */etc/netplan/00-installer-config.yaml* w *Ubuntu*.

```
# This is the network config written by 'subiquity'
network:
  ethernets:
    eth0:
      dhcp4: true
    eth1:
      addresses:
        - 10.100.0.10/24
  version: 2
```

W systemie *Windows* dla wirtualnej karty sieciowej podłączonej do wirtualnego przełącznika *Internal* ustawić adres IP 10.100.0.1/24.

Sieć *Internal* będzie służyła do komunikacji między hostem (*Windows*) a maszyną wirtualną (*Ubuntu*) (statyczne adresy IP).

W maszynie wirtualnej *Ubuntu* zainstalować: *Qemu*, *zip*.

```
sudo apt update
sudo apt install qemu-system-x86
sudo apt install zip
```

Ustawić hasło dla użytkownika *root* (będziemy łączyli się przez *WinSCP* na koncie *root*).

```
sudo apt passwd root
```

Zmodyfikować plik konfiguracyjny serwera *openSSH*, tak żeby *root* mógł się logować zdalnie- dodać do pliku konfiguracyjnego */etc/ssh/sshd_config* wpis:

```
PermitRootLogin yes
```

Do łączenia się z konsoli systemu *Windows* z serwerem *Ubuntu* możemy skorzystać z *ssh* łącząc się również jako *root*.

```
ssh wgrabski@10.100.0.10
```

Do przesyłania plików można skorzystać z programy *WinScp* do pobrania ze stony https://winscp.net/.

Utworzyć na serwerze *Ubuntu* w katalogu domowym katalog *minix* i pobrać do niego skrypt *minix.sh* wykonany przez byłego studenta, który ułatwia uruchamianie *minix*-a i montowanie obrazu dysku.

peku33/minix-toolkit: Skrypt do zautomatyzowanej pracy z systemem minix 2.0.3 na laboratorium SOI (github.com).

Ustawić do pobranego skryptu uprawnienie x dla użytkownika.

```
chmod u+x minix.sh
```

Teraz można uruchomić skrypt jako *root* (wymagane uprawnienia do montowania obrazu dysku).

Do pracy z plikami (po zamontowaniu obrazu przez skrypt) można skorzystać z programu *WinSCP* łącząc się z serwerem *Ubuntu* na koncie *root* (wtedy możemy modyfikować pliki na podłączonym obrazie dysku systemu *Minix*.

3 Mapowanie obrazu dysku systemu Minix

Po uruchomieniu skryptu minix.sh jako root

```
sudo ./minix.sh
```

katalog */usr* z obrazu dysku systemu *Minix* jest mapowany i dostępny w katalogu *minix_usr*. Obraz jest odmontowywany na czas uruchomienia systemu *Minix*.

4 Uruchomienie systemu Minix

Uruchamiamy skrypt *minix.sh* i wybieramy opcję uruchomienia systemu *Minix*.

W systemie *Minix* logujemy się na konto *root* bez hasła.

5 ZAMYKANIE SYSTEMU MINIX

W celu poprawnego zamknięcia (wyłączenia) systemu należy wydać polecenie *shutdown (p*roszę nie korzystać z poleceń: *restart* i *poweroff))* a następnie przechodzimy do konsoli qemu (Alt 2) i wpisujemy polecenie quit. Przełączanie pomiędzy witalizowanym systemem a konsolą qemu przy pomocy kombinacji klawiszy:

- *Alt 1* – witalizowany system,
- *Alt 2* – konsola *Qemu*.

6 KOMPILACJA JĄDRA

Po zmianie kodu źródłowego jądra należy przeprowadzić jego kompilację i instalację.

Przejść do katalogu:

```
/usr/src/tools
```

W celu kompilacji i instalacji nowego jądra wywołać:

```
make hdboot
```

W celu usunięcia wszystkich plików pośrednich wywołać:

```
make clean
```

6.1 ZMIANA ROZMIARU TABLICY DESKRYPTORÓW PROCESÓW

Ponieważ przy testowaniu wywołań systemowych może być konieczne utworzenie większej liczby procesów, w jądrze trzeba zwiększyć rozmiar tablicy deskryptorów procesów (inaczej nie można powołać więcej niż kilkanaście procesów, nie ma błędu przy tworzeniu procesów ale system nie działa prawidłowo).

Plik: */usr/include/minix/sys_config.h*

Zwiększyć wartość stałej z 32 na 64

```
/* Number of slots in the process table for user processes. */
#define NR_PROCS          64
```

6.2 DODANIE FUNKCJI REALIZUJĄCYCH WYWOŁANIA SYSTEMOWE

W pliku: */usr/src/mm/proto.h*

- Dodać prototyp funkcji realizujących wywołania systemowe

```
/* Function prototypes. */

struct mproc;
struct stat;

_PROTOTYPE( int do_maxChildren, (void) );
_PROTOTYPE( int do_whoMaxChildren, (void) );


/* alloc.c */
_PROTOTYPE( phys_clicks alloc_mem, (phys_clicks clicks)                );
```

- W pliku (może być inny plik z serwera MM): */usr/src/mm/misc.c*
- Dodać definicję funkcji realizującej wywołanie systemowe

```
/* Miscellaneous system calls.                    Author: Kees J. Bot
 *                                                31 Mar 2000
 *
 * The entry points into this file are:
 *   do_reboot: kill all processes, then reboot system
 *   do_svrctl: memory manager control
 */

#include "mm.h"
#include <minix/callnr.h>
#include <signal.h>
#include <sys/svrctl.h>
#include "mproc.h"
#include "param.h"

int childrenCount( int proc_nr )
{
    int children = 0;
    int i = 0;
    for (i = 0; i < NR_PROCS; ++i)
          if ((mproc[i].mp_flags & IN_USE) && proc_nr != i &&
(mproc[i].mp_parent == proc_nr))
                ++children;
    return children;
}

void maxChildren( int * children, pid_t * who )
{
    int maxChildren = -1;
    pid_t found = -1;
    int count = 0;
    int proc_nr = 0;
    for (proc_nr = 0; proc_nr < NR_PROCS; ++proc_nr)
    {
          if (mproc[proc_nr].mp_flags & IN_USE)
          {
                count = childrenCount( proc_nr );
                if (count > maxChildren)
                {
                      maxChildren = count;
                      found = mproc[proc_nr].mp_pid;
                }
          }
    }
    *children = maxChildren;
    *who = found;
}

PUBLIC int do_maxChildren()
{
    int children = -1;
```

```
    pid_t who = -1;
    maxChildren( & children, & who );
    return children;
}

PUBLIC int do_whoMaxChildren()
{
    int children = -1;
    pid_t who = -1;
    maxChildren( & children, & who );
    return who;
}
```

6.3 W SERWERACH MM I FS DODAĆ WPISY DO TABLICY WYWOŁAŃ SYSTEMOWYCH

W pliku: */usr/src/fs/table.c*

- Dodać wpisy dla dodawanych wywołań systemowych oznaczając je jako nie używane przez serwer FS

```
/* This file contains the table used to map system call numbers onto the
 * routines that perform them.
 */

#define _TABLE

#include "fs.h"
#include <minix/callnr.h>
#include <minix/com.h>
#include "buf.h"
#include "dev.h"
#include "file.h"
#include "fproc.h"
#include "inode.h"
#include "lock.h"
#include "super.h"

PUBLIC _PROTOTYPE (int (*call_vec[]), (void) ) = {
    no_sys,           /*  0 = unused    */
    do_exit,    /*  1 = exit      */
    do_fork,    /*  2 = fork      */
    //…
    do_svrctl,  /* 77 = SVRCTL */
    no_sys,           /* 78 = maxChildren */
    no_sys,           /* 79 = whoMaxChildren */
};
/* This should not fail with "array size is negative": */
extern int dummy[sizeof(call_vec) == NCALLS * sizeof(call_vec[0]) ? 1 :
-1];
```

W pliku: */usr/src/mm/table.c*

- Dodać wpisy dla dodawanych wywołań systemowych dla serwera MM

```
/* This file contains the table used to map system call numbers onto the
 * routines that perform them.
 */

#define _TABLE

#include "mm.h"
#include <minix/callnr.h>
#include <signal.h>
#include "mproc.h"
#include "param.h"

/* Miscellaneous */
char core_name[] = "core"; /* file name where core images are produced
*/

_PROTOTYPE (int (*call_vec[NCALLS]), (void) ) = {
    no_sys,           /*  0 = unused    */
    do_mm_exit, /*  1 = exit      */
    do_fork,    /*  2 = fork      */
    //…
    do_svrctl,  /* 77 = svrctl    */
    do_maxChildren,     /* 78 = maxChildren */
    do_whoMaxChildren,  /* 79 = whoMaxChildren */
};
/* This should not fail with "array size is negative": */
extern int dummy[sizeof(call_vec) == NCALLS * sizeof(call_vec[0]) ? 1 :
-1];
```

6.4 PROGRAM TESTUJĄCY DODANE WYWOŁANIA SYSTEMOWE MAXCHILDREN, WHOMAXCHILDREN

- Plik maxChildren1.c

```
#include <stdio.h>
#include <lib.h>
int main()
{
    message m;
    int ret = _syscall( MM, MAXCHILDREN, & m );
    printf( "Max children: %d\n", ret );
    return 0;
}
```

- Plik whoMaxChilderen1.c

```
#include <stdio.h>
#include <lib.h>
int main()
{
    message m;
    int ret = _syscall( MM, WHOMAXCHILDREN, & m );
    printf( "Max Children Parent: %d\n", ret );
    return 0;
}
```

- Plik maxChilderen.c

```
#include <unistd.h>

int main( int argc, char ** argv )
{
    int i;
    message m;
    int ret;
    int children;
    if( argc != 2)
          return 0;
    children = atoi(argv[1]);
    printf( "create %d children\n", children);

    for( i = 0; i < children; ++ i )
    {
          if( fork() == 0 )
          {
                sleep( 5 );
                return 0;
          }
    }
    sleep( 1 );

    ret = _syscall( MM, MAXCHILDREN, & m );
    printf( "syscall MAXCHILDREN return: %d\n", ret );
    ret = _syscall( MM, WHOMAXCHILDREN, & m );
    printf( "syscall WHOMAXCHILDREN return: %d\n", ret );
    return 0;
}
```