Operációs rendszerek gyakorlat terv

**1.gyakorlat. Alapok, szöveges függvények, fájlkezelés.**

A félév feladatait C-ben fogjuk megoldani, de használhatnak C++ -t is.

Kiszolgáló: oprendszerek.inf.elte.hui. A kódot vagy a szerveren szerkesztjük a vi,joe,mc segítségével, vagy ftp-vel a kliens gépről másoljuk át. A fordításhoz a cc vagy gcc használható. Mindkét link ugyanazt a gcc-4.3 programot hívja! A g++ parancs használatnál alapértelmezésben minden kódot C++ kódnak tekint, a g++ is a cc-t hívja! Futtatáshoz ne felejtsünk el futási jogot adni a programnak.

Javasolt feladatok a gyakorlatra:

1. „Hello világ” program készítése

Segítség:  
 include file-ok: #include <stdio.h>

fordítás: gcc fnev.c –Wall –o fnev (C program fordítása)

g++ fnev.cc –o fnev (C++ fordítás)

1. Mintafájl: strings.c, //char tömbök használata, saját hossz függvény

Segítség:  
 include file-ok:#include <string.h>

1. Mintafájl: args.c // Program paraméterek feldolgozása, paraméterek kiírása.

Segítség: a 0. argumentum a fájl neve  
include file-ok: -

1. Mintafájlok: open\_fopen.c, file.c //Fájlkezelés, szöveges, bináris. Struktúra írás

Segítség:  
 include file-ok: #include <stdio.h>, //fopen

#include <errno.h> //errno, perror

#include <sys/types.h> //open

#include <sys/stat.h> //open

#include <fcntl.h> //open

1. Mintafájlok: directory.c, passwd.c,mtime.c //Könyvtárkezelés, felhasználói azonosító meghatározása, fájl módosításának ideje

Segítség:  
 include file-ok: #include <sys/types.h> //opendir, readdir, getpwuid

#include <dirent.h> //opendir

#include <sys/stat.h> //stat, fstat

#include <pwd.h> //getpwuid, struct passwd

#include <time.h> //ctime

**Feladatok:**

* lefttrim függvény megvalósítása,karaktercsere, szövegkeresés
* struktúraírás, olvasás, (név és születési idő tárolása)
* ls –R megvalósítása, ls -al megvalósítása

**2.gyakorlat. Folyamatok létrehozása**

1. Mintafájl: fork.c //Gyerekfolyamatok létrehozása, fork(), wait(), waitpid(), getpid(), getppid(),

Segítség:  
 include file-ok: #include <unistd.h> //fork, getpid

#include <sys/types.h>//wait, getpid

#include <sys/wait.h>//wait

1. Mintafájl: exec.c // Külső program végrehajtása, az exec rendszerhívás család

Segítség:  
 include file-ok: #include <unistd.h> //exec

1. Mintafájl: system.c //Külső program hívása, system() rendszerhívás –

Segítség: különbség exec-kel (saját gyerek folyamatot hoz létre, abban cseréli le a tartalmat)  
 include file-ok: #include <stdlib.h> //system

1. Mintafájl: fcntl\_0, fcntl.c.c //Fájlok írása zárolás nélkül és zárolással , sleep(),usleep()

Segítség:

include file-ok: #include <unistd.h>//fcntl, sleep

#include <fcntl.h> //fcntl, open

#include <sys/stat.h> //open

1. Mintafájl: rand.c //Véletlenszámok generálása rand(), srand() függvény

Segítség: //srand(getpid()) –et használjuk minden folyamat elején, hogy minden folyamat különböző véletlenszámokat generáljon

include file-ok: #include <stdlib.h>//rand(),srand()

#include <sys/types.h>//getpid

#include <unistd.h>//getpid,fork

#include <sys/wait.h> //wait

**Feladatok:**

* Készítsünk 2 gyerekfolyamatot, amelyek 50 db 1-100 közötti véletlen számot írjanak ki a képernyőre és a saját PID-jüket. A szülő várja meg a gyerekfolyamatok befejeződését! (Véletlenszám rand(),srand() ) // rand,srand #include <stdlib.h>
* Saját shell készítése (execv vagy system)
* A gyerekfolyamat írja ki karakterenként a „Gyerek vagyok Gyerek vagyok…” szöveget, a szülő pedig olvassa ezt be, ha a gyerekfolyamat felengedte a zárolást.
* Készítsünk bináris fájlt, amelyik a neveket és a születési éveket tartalmazza – olvassuk is ezeket vissza.

**3.gyakorlat. Jelzések**

1. Mintafájl: signal.c //signal, kill rendszerhívások

Segítség:

include file-ok: #include <signal.h> //signal, kill

#include <unistd.h> //fork,getpid

#include <sys/types.h> //getpid

#include <sys/wait.h> //wait

#include <string.h> //strsignal – jelzés leírása

1. Mintafájl: sigaction.c

Segítség: //legtöbbször signal – lal egyenértékű (különbség, blokkolható jelzések beállítása, és SA\_FLAGS )

include file-ok: #include <signal.h> //sigaction, kill, sigemptyset,..

#include <unistd.h> //fork,getpid

#include <sys/types.h> //getpid  
#include <sys/wait.h> //wait

1. Mintafájl: sigprocmask.c //A jelzések blokkolhatóak, később megérkeznek sigprocmask, pause() – jelzésre vár

Segítség:

include file-ok: #include <signal.h> //sigprocmask

#include <unistd.h> //fork, pause

#include <sys/types.h> //wait

#include <sys/wait.h>//wait  
Időzítő

1. Mintafájl: setitimer.c //Időzítés

Segítség: //időzítő elindulásának kezdete is beállítható

include file-ok: #include <unistd.h> //alarm

#include <sys/time.h> //setitimer

#include <signal.h> //signal

**Feladatok:**

* Készítsen óra alkalmazást! (setitimer vagy alarm, time(),ctime(&time\_t), „\r” sor elejére áll)
* Készítsen alkalmazást két gyerekfolyamattal. A gyerekfolyamatok véletlenszámokat generálnak (1-100) és írnak ki, amíg 10-nél kisebb számot nem kapnak, ekkor küldjenek jelzést a szülőnek (mindegyik másikat), amelyik mindkét esetben írja ki azok fogadását.
* Készítsen alkalmazást 2 gyerekfolyamattal. Az egyik gyerekfolyamat küldjön jelzést (SIGUSR1) a szülőnek, aki írja ki, hogy megkapta és azonnal küldjön egy másikat (SIGUSR2) a másik gyereknek, aki fogadja azt és kiírja.

**4. gyakorlat. Csövek (névtelen, nevesített), cső állapot változás (select, poll rendszerhívás)**

1. Névtelen cső:

#include <unistd.h>

int pd[2]; pipe(pd); //Ha ez -1-el tér vissza hiba van.

pd[0] – a névtelen cső olvasás fájl azonosítója lesz

pd[1] – a névtelen cső írás fájl azonosítója lesz.

Jellemzően valaki ír, másik olvas, de lehet egyszerre mindkét műveletet is használni.

Mintapélda: pipe.c

Feladat: Írja be egyik folyamat a csőbe a parancs paramétereit, megfelelő elválasztással, míg a másik folyamat olvassa ki azokat!

2. Nevesített cső:

#include <sys/types.h>

#include <sys/stat.h>

int fd=mkfifo(„alma.ftc”, S\_IRUSR|S\_IWUSR); // -1 esetén hiba, 0-ok, további jogok:S\_IRGRP, S\_IWGRP

Ha valaki olvasásra nyitja meg a fájlt, az addig blokkolódik, amíg valaki írásra nem nyitja meg! Ez igaz névtelen csőre is.

Mintapélda: nevcso.c

Feladat: Írjon olyan programot, amiben a szülő-gyerek folyamatok beszélgetnek egymással, amíg egyik be nem fejezi a beszélgetést!

3. Állapot változás, select, poll rendszerhívás.

A select a paraméterül megadott fájlleírókat vizsgálja és addíg vár, amíg vagy változás történik bármelyik leíróban vagy az utolsó paraméter időzítő le nem telik.

#include <sys/select.h>

int result=select(f+1,&watchedpipes\_read,NULL,NULL,&tv);

1. paraméter: a max. fájlleírótól 1-el nagyobb érték.
2. paraméter: olvasásra váró fájlleírók
3. paraméter: írásra váró fájlleírók
4. paraméter: kivétel, hibaleírók
5. paraméter:timeval struktúra cím, időzítő, tv\_sec-másodperc, tv\_usec-mikrosec

A leírás szerint általános fájlleírókról szól, de valójában a „rendes” fájlokkal nem megy, csak csövekkel és sockettel.

Mintapélda:sel\_cso.c

A poll rendszerhívás a selecthez hasonló. A select a BSD unix családban jelent meg 1983-ban, majd a poll a SystemV unix része lett 1986-ban. Ma mindkettő POSIX standard és az oprendszerek Suse Enterprise rendszernek is része.

#include <sys/select.h>

int result=poll(poll\_fds,1,8000);

1. paraméter: pollfd struktúra tömb (poll\_fds). A pollfd struktúrának három mezője van. Az fd mező a fájlleíró, az events mező a POLLIN,POLLOUT stb. események bitenkénti vagy eredménye. A revents mezőben kapjuk meg azt a bitet, amilyen esemény az fd-vel kapcsolatban bekövetkezett. Ezt kell tesztelgetni, mikor a poll visszatért.
2. paraméter: A poll\_fds tömb elemszáma,megadja, hogy hány fájlleírót vizsgálunk.
3. paraméter: Időzítő érték millisec-ben! (a ppoll-ban az időzítő paraméter már timespec típusú (tv\_sec, tv\_nsec))

Mintapélda: poll\_cso.c

A selectnek, pollnak létezik pselect, ppoll testvére. Mindkét esetben egy sigset\_t mutatóval azt mondhatjuk meg, hogy milyen jelzésekre fejeződjön be az eseményre várakozás! A sigset\_t egy mask, tehát azokat a jelzéseket nem tartalmazza, amire a ppoll érzékeny.

Mintapélda: ppoll\_cso.c

Feladat:Írja át az előző beszélgetős feladatot select vagy poll használattal!

Feladat: A szülő folyamat elküldi egy térbeli pont koordinátáit a gyerekének csövön, míg az eldönti, hogy a pont egy gömb belsejében van-e? Ezt addig csinálják amíg meg nem unják!

**5.gyakorlat. Üzenetküldés (msg, mq), osztott memória**

1. System V üzenetcsalád

System V üzenetsor: int id msgget(kulcs, jog), kulcsot ftok generálja, vagy egy sima szám. Nem megfelelő kulcs esetén -1-et ad vissza msgget!!!! kulcs=ftok(argv[0],1);

Üzenet küldés: msgsnd( id, &uz,strlen ( uz.mtext ) + 1, 0 ); // 0==IPC\_NOWAIT

Üzenet fogadás: msgrcv( id, &uz, 1024, 5, 0 ); //5 == üzenet osztályazonosító

Üzenetsor vezérlés: msgctl( id, IPC\_RMID, NULL ); // törlés

Az üzenet egy struktúra, az első mezője egy long, üzenet osztályazonosító, a struktúra maradék része bármi lehet, jellemzően szöveg.

Mintapélda: uzenet.c

2. POSIX üzenetcsalád

Bár már a SystemV család örökségeként használható msg üzenetcsalád is Posix szabvány, az mq\_ üzenetsort tartják igazi Posix üzenetes rendszerhívásnak! Talán csak 1 lényeges különbség van, a mq\_notify hívás, aminek a segítségével egy folyamat értesítést kérhet ha üzenet érkezik a sorba, illetve a sor azonosítója a select, poll paramétere is lehet!!! Jelzést kap az a folyamat, aki meghívja a mq\_notify függvényt. Egy sorra csak 1 feliratkozás lehet!

#include <mqueue.h>

mqd\_t id=mq\_open(név, O\_CREAT|O\_RDWR, 0600,&attr); // attr.mq\_maxmsg és attr.mq\_msgsize értékeknek a rendszerértéknél kisebbnek kell lenni. (/proc/sys/fs/mqueue/ -ben vannak a rendszer maxok)

Fordítás: -lrt opcióval

Mintapélda: mq\_uzenet.c

3.Osztott memória

Létrehozása:

int oszt\_mem\_id=shmget(kulcs,500,IPC\_CREAT|S\_IRUSR|S\_IWUSR); //kulcs ftok-os, 500 bájt hosszú

Kapcsolódás:

char\* s = shmat(oszt\_mem\_id,NULL,0); // NULL a rendszer választ memóriát

A harmadik paraméter, flag legjellemzőbb értéke: SHM\_RDONLY, egyébként olvasásra, írásra kapcsolódik.

Lekapcsolás: shmdt(oszt\_mem\_id);

Törlés: shmctl(oszt\_mem\_id,IPC\_RMID,NULL);

Mintapélda: osztmem.c

Feladat: Küldjünk üzenetsoron keresztül két pont(sík) koordinátáit egy folyamatnak, majd az küldje vissza osztott memóriába a szakasz hosszát!

**6. gyakorlat. Szemaforok**

1. System V szemaforok

#include <sys/sem.h>

Létrehozása a semget paranccsal. Ez valójában egy szemafor családot hoz létre N szemaforral: int semid=semget(kulcs,N,IPC\_CREAT|S\_IRUSR|S\_IWUSR)

Ha -1 a semid, akkor sikertelen a léterhozás.

A szemafor család i. szemaforjának beállítása: semctl(semid,i,SETVAL,szemafor\_ertek)

Ha a semctl -1-et ad vissza, sikertelen a beállítás. Ha 1 szemafor van a családban, N=1, akkor i csak 0 lehet (i az index szám a szemafor tömbben)

Szemafor törlés: semctl(semid,0,IPC\_RMID), a második paraméter (0) nem érdekes, a teljes szemafor családot törli!!!

Szemafor művelet:

muvelet.sem\_num = 0; // szemafor index

muvelet.sem\_op = 1; // 1 up, -1 down

muvelet.sem\_flg = 0; // várakozik ha kell, lehetne IPC\_NOWAIT is!!

semop(semid,&muvelet,1);

A muvelet egy sembuf mutató, a semop harmadik paramétere megmondja, hogy muvelet tömbnek hány eleme van, hány szemafor műveletet kell elvégezni!

Mintapélda: szemafor.c

2. Posix szemafor család

A Posix szemafor utasítás család újabb, csak a 2.6-os kernelek után érhető el. Nem szemafor családot gyárt!

#include <semaphore.h>

Fordítás: cc –lrt pos\_szemafor.c

Létrehozása: Nevesített sem\_open(„alma”,..), a nem nevesítettet a sem\_init hozza létre, elsősorban szálakhoz kötődik.

sem\_t \*semid=sem\_open(„alma”,O\_CREAT,S\_IRUSR|S\_IWUSR,szemafor\_ertek );

if (semid==SEM\_FAILED) perror("sem\_open");

Törlés: sem\_unlink(semid);

Down: sem\_wait(semid);

A sem\_wait várakozik, ha a szemafor értéke 0. Létezik a sem\_trywait(semid) is, ez ha a szemafor értéke 0 akkor nem vár, hanem rögtön visszatér és beállítja errno értékét EAGAIN-re! A sem\_timedwait(semid, timeout) forma vár szemafor 0 esetén a timeoutban megadott ideig. Ha ezalatt nem lesz a szemafor szabad, akkor errno=ETIMEDOUT-ból láthatjuk , hogy a szemafor még blokkolt, de a türelmünk elfogyott.

Up: sem\_post(semid)

Mintapélda: pos\_szemafor.c

Feladat: Pék-Vásárló probléma megoldása. (pek\_vasarlo.c, termelo\_fogyaszto\_systemv.c)