Informazio-Sistemen Arkitektura

Telekomunikazio Teknologiaren Ingeniaritzako Gradua. 3. Maila.
Proba praktikoa

2016ko ekainaren 24a Iraupen osoa: 2½ h.

Gakoen erdia jaso behar dira proba gainditzeko

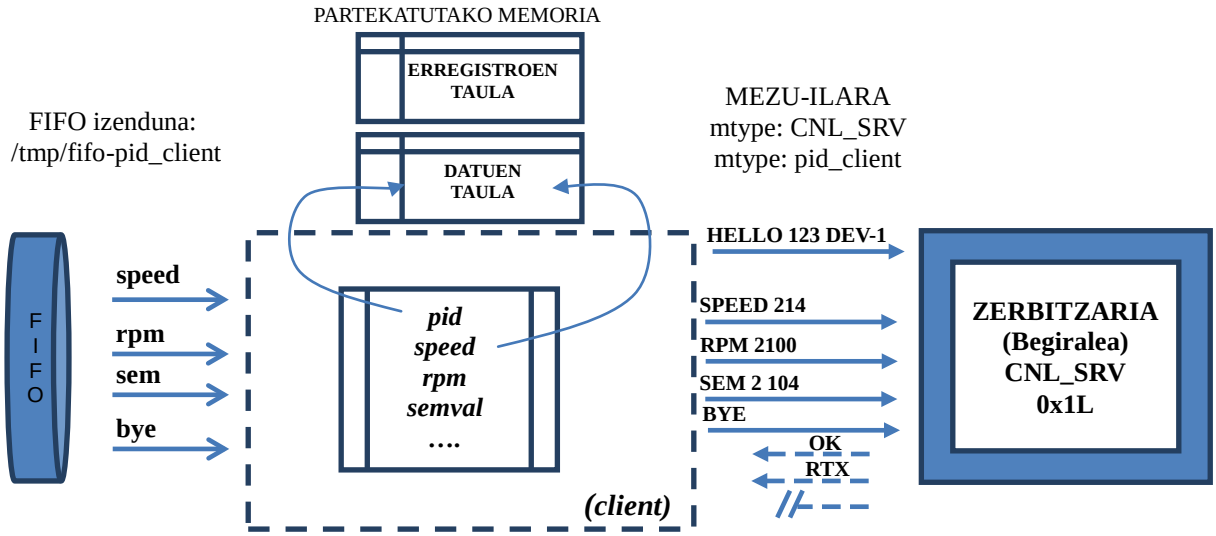
Proba praktikoaren azalpena

Baliabide guztiek, partekatutako memoriak, semaforoek eta mezu-ilarak ikaslearen NAN zenbakia erabiliko dute hamaseitarrean eta hizkirik gabe long formatuan (adib: NAN:11234567G eta gakoa 0x11234567L) bere burua identifikatzeko.

Begiraleak zerbitzari moduan lan egingo du sistemako mezu-ilara batetik mezuak jasoz. Mezu hauek CNL_SRV (0x1L) motakoak izango dira. Ikasleak garatu behar dituen client prozesuek kanal horretara bidaliko dituzte mezuak bere kontroleko aldagaien egoeraren berri emateko: abiadura (speed), minutuko birak (rpm) eta semaforoen egora (sem). Zerbitzariak bezero bakoitzari erantzun-mezua bidaliko dio mezu-ilararen bidez bezeroaren pid-arekin identifikatutako kanal edo mezu-mota erabiliz.

Ariketa hau modu inkrementalean garatuko da, pausu bakoitzean egindakoari funtzionalitate berriak gehituz eta atzeranzko bateragarritasuna mantenduz. Hau da, 2. ariketa 1. ariketaren hobekuntza izango da. Eta horrela, 3. ariketan funtzionalitate guztiak (gako guztiak) bildu arte.

Sistemaren arkitektura ondoko irudikoa da:



Erregistroen taula partekatutako memoriako segmentu batean dagoen array bat da. Erregistroek sisteman identifikatutako gailu bakoitzaren egoera gordetzen dute. client programa HELLO komandoaren bidez identifikatu beharko da zerbitzarian eta ondoren, "/tmp/fifo-pid_client" FIFO ilaratik etorriko diren mezuen zain geratuko da gailuen informazioa lortzeko (*speed*, *rpm* eta *sem*). Jasotako mezuetan datorren informazioa zerbitzariari bidaliko dio gailuen parametroen berri emateko. Parametro hauek gailuari dagokion datuen taulako sarrera egokian idatzi beharko dira partekatutako memorian. client bakoitza zerbitzarian identifikatzen denean, zerbitzariak erregistroen taulako sarrera bat gordeko dio. Erregistro horren indize bera datuen taularako erabili beharko da saioko datuak bertan gaurkotzeko.

Saio desberdinak ezarri daitezke konkurrenteki terminal desberdinetatik, baina horretarako gailu bakoitza izen desberdin batekin konektatu beharko da. Adibidez, terminal batean $./client device1 eta bestean $./client device2 jarriz. Zerbitzariak saioak etengo ditu epe luze batean (15 segundo) informaziorik jaso ezean.

1. Ariketa. Oinarrizko komunikazioa

Lehen ariketa honetan oinarriko komunikazioa ezarri behar da, identifikatuz, datu bat bidaliz, eta deskonektatuz. Zerbitzaria mezu-ilarako CNL_SRV (0x1L) kanalean adi egongo da. Espero dituen mezuen formatua ondokoa da:

<mtype (long=0x01)><pid (int=pid_client)><komandoa (komandoaren testua)>

Komunikazio protokolo hau mezuen trukean oinarritzen da eta irakurgarriak diren testuak garbian igortzen ditu. Hasierako komandoak honakoak dira:

1.- HELLO pid Izena (Adib: HELLO 2134 DEVICE-1)

2.- BYE

Komando hauei beti erantzungo zaie OK eta iruzkin batekin emaitzaren balorazioaz. BYE salbuespena da; honek ez du erantzunik itxarongo eta programa berehala amaituko du.

- OK iruzkina // Iruzkinean komandoari buruzko informazioa etor daiteke

1 gakoa konexioa HELLO mezu-formatu egokiaz lortutakoan jasoko da. Begiraleak HELLOan etorri den pid zenbakia duen prozesuak bizirik dirauela ikusitakoan gailua erregistratuko du eta 2 gakoa erakutsiko digu. client prozesuak FIFO ilara ondo sortu badu 3 gakoa erakutsiko da. FIFOaren izena "/tmp/fifo-pid_client" izan beharko da (non pid_client exekuzioan dagoen client prozesuaren identifikatzailea den). Begiralea fifoan idazteko gai bada, gakoa erakutsiko du. 4 gakoa BYE komando bidali eta berehala saioa eteten irtenez gero agertuko da.

2. Ariketa. Komunikazio OSOA errorerik gabe.

Ariketa honetan dauden komando guztiak erabiliko ditugu komunikazio oso bat egiteko. Saio bakoitza HELLO komando bat datu egokiekin bidaliz hasiko da.

Ondoren prozesu zenbakiarekin sortutako fifotik (Adib: client prozesuaren pid zenbakia bada /tmp/fifo-1232 fifotik) egoera-parametroak jasotzeko zain geratuko da gure programa. Gailuen egoerari buruzko informazioa (speed, rpm, etab…) fifo izendunetik jasoko du client-ek. Fifotik jasotako mezuek ondoko formatua dute:

<Byte char motakoa><parametroak komandoaren arabera >

Fifoko mezu-motak eta formatuak honakoak dira:

- SPEED: ('2')(<abiadura **float** formatuan>) Adib: 2 127.56
- RPM: ('3') (RPM balioa **ASCII** formatuan) Adib: 3<324>
- SEM: ('4') (semaforo zenbakia **int** bitarrean) Adib: 4 2
- BYE komandoa: ('5') Adib: 5

Mezu hauek jasotakoan, zerbitzariari igorriko zaizkio komando egokiak. Ondokoak dira komando horiek:

- SPEED (abiadura **ascci** formatuan) Adib: SPEED 127.56
- RPM (birak **ascci** formatuan) Adib: RPM 342
- SEM (zenbakia **ascci**) (balioa **ascci**) Adib: SEM 2 109
- BYE

5 gakoa SPEED komando edo RPM komando egokiekin lortuko da. 6 gakoa biak betez gero. 7 gakoa 4 semaforo dituen arraitik zenbakiari dagokion semaforoaren balioa zuzena bada. 8 gakoa, konkurrenteki bi saio mantentzen badira.

Partekatutako memoriaren (SIZE:1024) 0 posizioan 4 erregistrodun taula bat (array bat) gordeta dago (*struct st_reg*) egitura duena. Partekatutako memoriaren 300 posizioan 4 erregistrodun beste taula bat dago (*struct st_data*). Bigarren taula honetan saio bakoitzaren parametroak gordetzen dira. Adibidez, saio bat erregistro-taulako 2. erregistroan aktibatu bada, indize bera erabiliko da datu-taulako 2. sarreran datuak gordetzeko.

client programak "name" eremua erabiliz, 4 erregistroetatik edozeinetan egon daitekeen saio aktiboa bilatu beharko du indizea lortzeko, eta indize bera erabiliz, datuen taulako edukia egokitu beharko du beste taulan. Zerbitzariak ausaz erabakiko du balio bat saio osorako.

Speed edo rpm gaurkotzen badira 9 gakoa lortuko da. Biak gaurkotuz gero, 10 gakoa. Kasu honetan, aukeratutako semaforoaren balioa irakurtzeko 0 semaforoa (arraiko lehena) erabiltzen badugu atal kritikoa egiteko, 11 eta 7 gakoak lortuko dira. 12 gakoa bi saio konkurrenteri dagozkien pid-ak gaurkotzen badira ($./client device1 eta $./client device2 terminal desberdinetan), bakoitza bere erregistroan.

3. Ariketa. Saio OSOA galerekin

Komunikazio batean mezuak galtzeko arriskua dago. Baita ere birtransmititzeko premia erantzunik ezean tenporizadoreak iraungitakoan. Ikasleak prozedura hauek inplementatu beharko ditu protokoloan. Ariketa honetan Begiraleak mezuen galerak simulatuko ditu ausaz bezeroak protokoloa nola inplementatu duen aztertzeko.
Bezeroaren komando baten aurrean zerbitzariak modu desberdinetan joka dezake:

- OK iruzkina //komandoa zuzen jaso denean
- RTX iruzkina //azken komandoa birtransmititzeko eskaera
- Erantzunik ez //handik **4 segundura** bezeroak berriz errepikatu mezua

Begiraleak saio bakoitzari birtransmitisio (RTX) eta mezu-galeren prozedurak simulatuko ditu. Bezeroak ondo erantzuten badie lehenengo saioko birtransmisio (RTX) eskaerei 13 gakoa jasoko du. Saio konkurrenteetan eginez gero, 15 gakoa. Tenporizazio bidez birtransmititzen badu, 14 gakoa, eta saio kokurrenteetan bada, 16 gakoa.

Oharra:
Tenporizazio bidezko birtransmisio prozedura inplementatzeko seinale tenporizatu bat erabiltzea proposatzen da (alarm(4)) irakurketa blokeagarri bat eteteko. Sistemaren azken inplimentazioaren zenbait dei ez dira ondo eteten irakurketa blokeagarrietan maskaratze-funtzioak egin ezean. Arazo hau ekiditeko ohiko signal()-en ordez signal_EINTR() funtzioa erabili. Horrela irakurketa blokeagarriak eten ahal izango ditugu denbora agortutakoan.

4. Ariketa. Saioak memorian sortu.

Gerta daiteke zerbitzaria erabilgarri ez egotea, kasu honetan sistemaren gailuek beren saioak sortu ditzakete partekatutako memoria-segmentuko erregistroetan. Memoriaren 0 posizioan 4 erregistro (*struct st_reg* motakoak) dituen taula bat dago sistemaren saioak gordetzeko. Arraiko 3. semaforoa (laugarren balioa) saioen taulara atzipen esklusiboa lortzeko erabiliko da, horrela erregistroak modu seguruan idazteko.

Ariketa hau aurreko hiru ariketen alternatiba bezala pentsatuta dago. Lehenengo hiru ariketak eginez gako guztiak lor daitezke. Ikasleak mezu-ilarako protokoloaz arazoak baditu, ariketa alternatibo hau eginez 8 gako desberdin lor ditzake (saioaren hasierakoak eta memorian datuak gaurkotzearenak). Zenbait gakok saio konkurrenteak eskatuko dituzte.

Bezeroak libre dagoen erregistro bati alta eman beharko dio. Horretarako bere egoera aldatu beharko dio (ST_FREE egoeratik) 1-era pasetzeko (ST_PID egoerara) eta erregistroan pid eta name eremuak betez. Name eremua 16 byte luze den arren, ahal den neurrian 8 karaktere bakarrik erabili. Begiraleak saio berriak dauden aztertuko du eta dena ondo badago, “/tmp/fifo-pid_client” izendun fifora gailuei dagozkien mezuak bidaliko ditu, bezeroak prozesatu ditzan eta memorian aurreko ataletan eskatu den bezala idatzi ditzan.

Informazio-Sistemen Arkitektura

Telekomunikazio Teknologiaren Ingeniaritzako Gradua. 3. Maila.
Proba praktikoa

2016ko ekainaren 24a Iraupen osoa: 2½ h.

Gakoen erdia jaso behar dira proba gainditzeko

Izena:……
Taldea:……...…… NAN:…….
Gela:……. Ilara:…….. Zutabea:….....SINADURA:

1 Ariketa

1 Gakoa:	2 Gakoa:	3 Gakoa:	4 Gakoa:

2 Ariketa

5 Gakoa:	6 Gakoa:	7 Gakoa:	8 Gakoa:

9 Gakoa:	10 Gakoa:	11 Gakoa:	12 Gakoa:

3 Ariketa

13 Gakoa:	14 Gakoa:	15 Gakoa:	16 Gakoa:

4 Ariketa

Ariketa honek aurreko 3 ariketetan lortu daitezkeen 16 gakoetatik zenbait gako lortzen ahalbidetzen du partekatutako memoria ondo erabiliz gero. 2,4,5,6,9,10 gakoak lortzen dira saio on batekin, eta 8 eta 12 gakoak saio konkurrenteekin.

Erreferentziarako client-ref.c erabili arikitetarako zenbait datuen espezifikazioekin arazorik ez izateko. Hemen doaz memoriako erregistroen taularako egiturak saioetarako (memoriako 0 posizio erlatiboan) eta dagozkien datuak gordetzeko datu-taula (300 posizio erltiboan):

```
#define ST_FREE 0
#define ST_PID 1
#define ST_DATA 2
#define LEN_NAME 16     //ahal den neurrian 8 byteko izenak erabili

#define OFF_REG_TBL     0
#define OFF_DATA_TBL    300

struct st_reg {
        int state;      // State of register
        char name[LEN_NAME];    // Name of device
        pid_t pid;      // Process identifier
};
struct st_data {
        float speed;    // Speed in float format
        int rpm;        // RPM as int
        int sem;        // Sem number
        int semval;     // Sem value
};
```