eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

Ingeniaritza Informatikoko Gradua

1.Zatia : Sistemaren Arkitektura

Mikel Laorden Gonzalo

Sistema Eragileak

Irakaslea: Ander Soraluze Irureta

Aurkibidea:

Helburua:

Lan honen helburua, sistema eragile baten kernelaren osagai nagusiak zeintzuk diren ikastea da. Horretarako honen osagai eta funtzionalitate batzuk garatuko dira kernel sumulatzaile bat sortuz. Funtzionalitate hauek, prozesuen kudeaketa eta planifikazioa, memoria-sistema eta sarrera/irteera izango dira.

Lan hau 5 zati ezberdinetan aurkeztuko da.

Sarrera:

Lehenengo zati honetan, sistemaren arkitektura zehaztuko da. Bertan, oinarrizko funtzionalitateak aparte, hauek inplementatzeko behar diren datu egiturak ere definituko dira.

Eskakizun orokorrak

Proiektuaren zati hau garatzeko bete behar diren eskakizunak hurrengoak dira:

- Proiektua C lengoaian egin beharko da.
- Simulatzailea konfiguragarria izango da. Parametroak jasoko ditu eta hauen arabera konfiguratuko da.
- Klasean ikusitako sinkronizazio elementuak erabili beharko dira
- Derrigorrezkoa izango da baldintzazko mutex-ak erabiltzea sinkronizazio eta komunikazioa burutzeko.

Inplementazio eskakizunak:

Hurrengo ataletako elementuak inplementatuak edota diseinatuak izango dira zati honetan:

- Hariak:
    - Erlojua: Tick-ak sortzen ditu
    - Tenporizadorea: Maiztasun finko bateki seinale bat sortzen du
    - Prozesuen Sortzailea: Maiztasun finko batekin prozesuak sortzen ditu
    - Scheduler/Dispatcher: Prozesuen planifikazioa eta testuingurualdaketak egiten ditu
- Datu egiturak:
    - Prozesuen Ilara: Sortu diren prozesu berriak mantentzeko
    - CPUak: CPU-ak, core-ak eta hariak mantentzeko datu egitura
    - Process Control Block (PCB) sinplea: Bakarrik Process ID (PID)-a izango duena.
- Funtzionamendua:
    - Erlojua: Sistema kontrolatzen duten zikloak sortzen ditu
    - Tenporizadoreak: Bi tenporizadore. Seinale bat sortuko du erlojuaren frekuentzia oinarritzat hartuz. Erlojuaren frekuentzia zatituz funtzionatuko du
        - Schedulerrari deitzeko
        - Prozesu sortzailea deitzeko

- Prozesu sortzaileak, prozesu berrien PCB sinpleak sortuko ditu. Hau da, bakarrik PID-a dutela.

Inplementazioa:

Atal honen inplementazioa egiteko, kodea antolatu egin da. Hainbat fitxategi sortu dira. Hauetako bakoitza garatu beharreko funtzionalitate, kontzeptu edo modulu batean oinarritutu da. Fitxategi bakoitzean, horri dagokion funtzionalitatea aurrera eramateko inplementatu behar den logika garatu da.

Era honetan, zerikusirik ez duten inplementazioak banatu dira kodearen berrerabilpena eta antolaketa sustatuz. Hurrengo ataletan, sortu diren moduluak aurkeztuko dira:

Erlojua eta Tenporizadorea:

Erlojua garatzeko 2 fitxategi sortu dira:

- erlojua.c (sheduler_proj/src/erlojua/erlojua.c): Erlojua eta Tenporizadorearen inplementazioa
- clk_s.h (sheduler_proj/include/data_structures/clk_s.h): Erlojua eta Tenporizadorearen definitzioa

clk_s.h

Erlojua eta tenporizadoreak martxan jartzeko behar diren funtzioak eta datu egiturak definitu dira. Hauek dira definitutako elementuak:

- **struct timer_s:** Tenporizadorea definitzeko struct-a. Bertan, lotuta duen erlojuaren mutex-aren erreferentzia, tenporizadorearen lan frekuentzia, Tick bat bidali nahi duen momentuan exekutatu behar duen funtzioaren erreferentzia eta aurreko funtzioa exekutatzerakoan honi gehitu behar zaion parametroaren erreferentzia.
- **struct clk_mutex_s:** Erlojua definitzeko struct-a. Bertan, erlojuaren mutex-a, tenporizadoreak eta erlojua bera sinkronizatzeko behar diren mutex aldagai baldintzatuak, erlojuari lotuta dauden tenporizadore kopurua eta uneko erloju tick-a jada kontsumitu duten timer kopurua (done aldagaia).

- **void *timer_start(void *tmr_struct) :** Tenporizadorea, pthread batean exekutatzeko funtzioa. Parametro bezala "struct timer_s" motako struct bat jasoko du.
- **void *clock_start(void *clk_struct) :** Erlojua, pthread batean exekutatzeko funtzioa. Parametro bezala "struct clk_mutex_s" motako struct bat jasoko du.

erlojua.c

Fitxategi honetan, "clk_s.h" fitxategian definitutako funtzioak definitzen dira. Funtzio hauek jada klasean landu ditugu, hortaz ez dira azalduko. Diseinu honetan, tenporizadoreak bidali behar duen sinkronizazio "seinalea" ez da mutex-en bitartez kudeatzen, funtzio deiak erabiliz baizik. Horregatik timer bakoitzak funtzio baten erreferentzia izango du.

Scheduler/Dispacher eta Prozesu Sortzailea:

Scheduler, Dispacher eta Prozesu sortzailea sortzeko hurrengo bi fitxategiak erabili dira:

- scheduler.c (sheduler_proj/src/erlojua/scheduler.c): Funtzioak inplementatu
- sheduler_s.h (sheduler_proj/include/data_structures/sheduler_s.h): Funtzioak eta Datu egiturak definitu

sheduler_s.h

Fitxategi honetan, aurreko funtzionalitateak garatzeko behar diren datu egiturak eta funtzioak definitu dira. Hauek dira definitutako elementuak:

- **struct sched_egitura**: Scheduler/Dispacher-aren egoera gordetzeko struct-a. Bertan unean exekutatzen ari den prozesua, prozesuen ilara eta prozesuen ilararen atzipena kotrolatzen duen mutexa osatzen dute.
    - "**sched_basic_t**" bezala ere ezagutzen da.
- **struct pr_gener_egitura**: Prozesuak sortzeko erabiliko den konfigurazioa eta egoera gordetzen duen egitura. Honek, prozesu sortzaileak sortu ditzazkeen PID máximo eta minimoak, kontadore bat (etorkizunean PID-ak sekuentzialki gehitzeko erabili ahal izango dena) eta scheduler struct bati (sched_basic_t motakoa) erreferentzia (era honetan, prozesu sortzaileak schedulerrari prozesu bat gehitzea eskatu ahal izango dio).

- **void sched_AddToProcessQueue(sched_basic_t * sched, pcb_t * new_pcb):** Scheduler egitura bat eta PCB egitura bat emanda, PCB hau prozesuen ilaran gehitzen du konkurrentziarekiko era seguruan.
- **void sched_Schedule(sched_basic_t * sched)**: Scheduler egitura bat emanda, hurrengo prozesu bat aukeratu eta exekutatzen du. Hau dispatcher funtzioaren laguntzaz burutzen du.
- **pcb_t * sched_GetNextFromProcessQueue(sched_basic_t * sched)**: Scheduler egitura bat emanda, exekutatu beharreko hurrengo prozesua erabaki eta hau prozesu ilaratik kentzen du.
- **void disp_Dispatch(pcb_t * old_pcb, pcb_t * new_pcb)**: Dispatcher-aren funtzioa. Aterako den prozesua eta sartuko dena eskainiz, testuinguru aldaketa burutzen du. Zati honetan, funtzio hori ez du ezer ez egingo.
- **sched_basic_t * sched_struct_create_and_init():** Scheduler egitura bat sortu eta hasieratzen du.
- **int _prgen_getValidPid(pr_gen_t * pr_gen):** Prozesu sortzaile baten egitura emanda, egokia den PID bat sortzen du. Funtzio hau ez dago guztiz amaituta. Ez da frogatzen sortu berri den PID-a duen beste prozesu bat jada existizen den ala ez.
- **void prgen_init(pr_gen_t *prgen, sched_basic_t * sched, int max_PID, int min_PID):** Prozesu sortzailearen konfigurazio parametroak eskainiz, prozesu sortzaile egitura bat hasieratzen du.
- **void prgen_generate(pr_gen_t * prgen);** : Prozesu sortzaile egitura bat emanda, honek banean duen konfigurazioa kontuan hartuz, prozesu berri bat sortu eta
- **void sched_print_sched_State(sched_basic_t * sched)** : Scheduler-aren egoera pantaiaratzeko erabilitako funtzioa. Prozesuen lista, prozesu kopurua eta unean exekutatzen hari den prozesuaren PID-a pantaiaratzen du.

sheduler.c

sheduler_s.h fitxategian definitutako funtzioak inplementatzen dira.

CPU egitura eta hauen funtzionalitateak:

Cpu egitura definitzeko hurrengo fitxategiak erabili dira:

- cpu_data_s.h (sheduler_proj/include/data_structures/cpu_data_s.h): Funtzioak eta datu egiturak definitu.
- cpu_fun.c (sheduler_proj/src/datu_egiturak/cpu_fun.c): Funtzio nagusiak inplementatu.

cpu_data_s.h

Hurrengo puntuetan CPU-aren egitura definitzeko datu egiturak eta oinarrizko funtzioak definitu dira.

- **struct cpu_s**: CPU baten oinarrizko egitura defintizen du. Haren barnean dauden core kantitatea gordetzen duen aldagaia eta core egituren (struct core_s) erreferentzien array bat du.
- **struct core_s**: Core baten oinarrizko egitura. Haren barnean, honek dituen hari kantitate kopurua eta hari egituren erreferentzien array bat du.
- **struct core_hari_s**: Core hari baten oinarrizko egitura. Bertan bakarrik, exekutatzen ari den prozesuaren PID-a gordetzen da. Hau, etorkizunean PCB erreferentzia eta hainbat registroengatik ordezkatuko da.

- **int cpu_t_init(struct cpu_s * p_cpu, int core_num, int hari_num_per_core)**: Parametro bitartez zehaztutako CPU egitura bat hasieratzen du zehaztutako core kopuruarekin eta core hauek dituzten hari kopuruekin.
- **int core_t_init(struct core_s * p_core, int hari_num)**: Core egitura bat emanda, core hau hasieratzen du zehaztutako hari kopuruarekin.
- **int hari_t_init(struct core_hari_s * p_hari)**: Hari egitura bat emanda, hari egitura ha hasieratzen du beharrezkoak diren eremuak aldatuz;

cpu_fun.h

cpu_data_s.h fitxategian definitutako funtzioen inplementazioa. Funtzio hauen inplementazioak, definizioan zehaztu dena egiteko sortu dira.

Oinarrizko datu egiturak: Lista estekatua

First-Last lista estekatu orokor bat sortu da. Lista hau sortzeko beharrezkoak izan diren nodoak ere definitu dira.

Lista honek, hasierako nodoari eta amaierako nodoari erreferentzia egiten duen aldagai bat izango du. Nire kasuan, Reversed First Last Linked List (LFRL) baten antzera jokatzeko programatu dut, queue bat bezala jokatzeko. Era honetan, listaren amaieratik (errealitatean listaren burua dena) elementu bat kentzea eta listaren hasierara (errealitatean listaren buztana dena) elementu bat gehitzea kostu konstantea izango du. Lista hau FIFO

politika eta etorkizunen batean, (linked list hauek array-ekin konbinatuz) priority queue politikak inplementatzeko erabili daiteke.

Elementu hau, hurrengo fitxategietan definitu da. Fitxategi hauetan dagoen kodea zehazki asko dokumentatu zen. Hortaz, haien funtzionamendua ulertzeko fitxategi hauei begirada bat eman:

- lnklist_s.h (sheduler_proj/include/data_structures/lnklist_s.h): Datu egituren eta funtzioen definizioa.
- lnklist_LFRL.c (sheduler_proj/src/datu_egiturak/lnklist_LFRL.c): Reversed First-Last Linked List datu egitura maneiatzeko funtzioak definitzen dira.

lnklist_s.h

Hauek dira fitxategi honetan definituta dauden elementuak:

- **struct lnk_node**: Onarrizko nodo egitura baten definizioa. Nodo honetan hurrengo nodoaren punteroa eta edozein motako datuetara punteroa egongo da definituta.
- **struct lnklist_fl**: Oinarrizko First-Last Linked List baten egitura. Struct honek, listaren lehenengo eta azkenengo nodoen erreferentzia eta listan dauden nodo kopurua gordeko da.
    - "**lnklist_LFRL**" bezala ere erreferentziatua izan daiteke.

- **void *lnklst_LFRL_pop(lnklist_LFRL * p_list)**: Parametro bitartez jasotako LFRL listatik lehenengo elementua (sartu den azkena) atera eta bueltatuko du.
- **void lnklst_LFRL_push(lnklist_LFRL * p_list, void * p_data)**: Parametro bitartez jasotako LFRL motako lista estekatuan, zehaztutako elementua gehitzen du. Horretarako nodo berri bat sortu eta listaren amaieran gehitzen du.
- **void lnklst_LFRL_init(lnklist_LFRL * p_list)**: Parametro bitartez pasatutako Reversed First-Last Linked List-a hasieratzen du.

lnklist_LFRL.c

Reversed First-Last Linked List-aren inplementazioa.

Process Control Block (PCB):

PCB-aren oinarrizko egitura pcb_s.h (sheduler_proj/include/data_structures/pcb_s.h) fitxategian definitu da. Bertan hurrengoa zehazten da:

- **struct pcb_str**: PCB-aren oinarrizko egitura. PID-a gordetzeko sarrera bat du.
    - "**pcb_t**" bezala ere erreferentziatu daiteke.

Programa nagusia:

Programa nagusian main funtzioa definitzen da. Bertan, parametro batzuk tukatuz, sortutako hasiera bateko kernel-aren funtzionalitatea aldatu daiteke. Programa hau bakarrik fitxategi batean aurkitzen da definituta zein inplementatuta, main.c (sheduler_proj/src/main.c) fixategian hain zuzen ere.

main.c

Atal honetan, programa honek burututako funtzioak laburbilduko dira, egiten duena era orokor batean ikusi daitezen.

1) Programaren main funtziotik kanpo hurrengo aldagaiak eta konstanteak hasieratzen dira:
    a) Erlojuaren datu egitura (clk_mutex_s) eta dituen mutex-ak zein aldagai kondizionalak ere

```
pthread_mutex_t g_mutex_clk = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t g_cond_clk_ziklo = PTHREAD_COND_INITIALIZER;
pthread_cond_t g_cond_tmr_arreta = PTHREAD_COND_INITIALIZER;

struct clk_mutex_s clkm_1 = {
        .mutex_clk = &g_mutex_clk,
        .cond_clk_ziklo = &g_cond_clk_ziklo,
        .cond_tmr_arreta = &g_cond_tmr_arreta,
        .tmr_kop_tick_consumed = 0,
        .tmr_total_kop = TIMER_KOP
    };
```

    b) Konfigurazio aldagaiak:
        - Sortuko diren 3 tenporizadore instantzien frekuentzia eta tenporizadore kopurua konstante bezala

```
#define TIMER_KOP    3
#define SCHEDULER_FREQUENCY   10
#define PROCESSGEN_FREQUENCY   5
#define EGOERA_PRINT_FREQUENCY 5
```

        - Prozesu sortzailearen konfigurazio aldagaiak:

```
#define PRGEN_MAX_PID 1
#define PRGEN_MIN_PID 1000
```

2) Programaren main funtzioan, sheduler-aren (sched_basic_t) eta prozesu sortzailearen (pr_gen_t) egiturak sortu eta hasieratuko dira.

```
sched_basic_t * my_scheduler;
my_scheduler = sched_struct_create_and_init();

pr_gen_t my_pr_generator;
prgen_init(&my_pr_generator, my_scheduler, PRGEN_MAX_PID, PRGEN_MIN_PID);
```

3) Ondoren harien aldagaiak defintzen dira. Hauek 4 izango dira: Hari hauek prozesu sortzailearen tenporizadorea, scheduler-aren tenporizadorea, erlojua eta sheduler-aren egoera pantaiaratzen arduratuko den tenporizadorea.

```
pthread_t tr_clk, tr_tmr1, tr_tmr2_prgen, tr_tmr_print;
```

4) Tenporizadore hauetako bakoitzarentzako tenporizadore egitura (timer_s) bat sortuko da. Bertan, tenporizadore bakoitzari dagokion frekuentzia, timer-tick oro exekutatu beharko duen funtzioa eta aurreko funtzio honen parametroak zehaztuko dira. Guztiek erloju berari konektatuta egongo dira.

```
struct timer_s tmr1 = {
    .linked_clk = &clkm_1,
    .tmr_tick_freq = SCHEDULER_FREQUENCY,
    .TickAction=&sched_Schedule,
    .tickActionParams=my_scheduler};
```

5) Azkenik, tenporizadoreak eta erlojua martxan jarriko dira, bakoitza hari ezberdin batean.

```
if (pthread_create(&tr_tmr1, NULL, timer_start, (void *) &tmr1)){
    printf("Error: Timer started\n");
}
```

Exekuzioa eta konpilazioa:

Programa osoa era erraz batean konpilatzeko makefile bat sortu da [1]. Make file honen bitartez build karpeta bat sortu eta bertan, beste hainbat fitxategi aparte, a.out fitxategi konpilatua sortzen du.
Konpilazioa, debugarekin bateragarria izateko (gcc … -g) diseinatu da.
Proiektua exekutatzeko orduan:

1. Proiektuaren erroan jarri eta make komandoa exekutatu
2. sheduler_proj/build/a.out fitxategia exekutatu

Ondorioak

Lan hau egiteko, c programazio lengoaia, ez zerotik baina bai maila oso oinarrizko batetik, ikasi egin behar izan da. Gainera, programa eraginkor eta hedakor bat sortzeko programazio diseinuan ere denbora inbertitu da. Nire partida puntua, jada ezagutzen ditudan objektuetara oinarritutako programazio lengoaiak izan dira. Ondorioz, proiektua objektuetara bideratutako programazio baten ikuspuntutik bideratu dut, c-ko struct-ak eta funtzioak Javako klaseak eta metodoak direlakoan analogia bat eginda hurrenez hurren.

Proiektu honen izaera aldakorraren ondorioz, hedakortasunean eta kodearen berrerabilpenean zentratu naiz. Era honetan, proiektu honetan egin beharreko aldaketak eta hobekuntzak egikaritzeko erreztasuna lortu nahi da.

Egindako lanak garatu beharreko hutsuneak ditu, hauek 3 motetan sailkatu daitezeke: Kritikoak, erabilgarriak eta bestelakoak

- Kritikoen inguruan:
    - Prozesu sortzailearen auzazko PID-aren esleipenak arazoak eman dezake 2 prozesu-ei PID berdina esleituz. Hau, sortutako PID-a une horretan erabiltzen ari den frogatzen ez delako gertatu daiteke. Errore hau, proiektuaren 2. atalean landuko da, sheduler-aren politikarekin bat datorren prozesu ilara egoki bat definitzen denean.
- Erabilgarriak:
    - Tenporizadore baten frekuentzia aldatzeko funtzioa. Honek, scheduler-aren quantum-a kontrolatzeko modu bat izan daiteke.
    - Beste tenporizadore bat erloju berdinari gehitzeko funtzioa.

- Bestelakoak:
    - Erroreen eta salbuespenen tratamendu estandarizatua
    - Proiektuaren fitxategien arteko dependentziak eta egitura optimoa
    - Funtzioen eta datu-moten izendatze arauak argitu
    - Programazio ingurunearen konfigurazio hobetu

Egin beharreko hauek ebazteko orduan, errore kritikoek lehentasuna izango dute eta proiektuaren azken zatirako ebatzita izango ditut. Denbora izatekotan, aurretik aipatutako bestelako atazak tratatzen saiatuko naiz.

Bibliografia:

[1]: A Super-Simple Makefile for Medium-Sized C/C++ Projects