Ingeniaritza Informatikoko Gradua

2.Zatia : Planifikazioa

Mikel Laorden Gonzalo

Sistema Eragileak

Irakaslea: Ander Soraluze Irureta

Inplementazioaren fitxategiak
https://github.com/mikelvin/SE_Proiektua

Aurkibidea:

1. Helburua

Proiektuaren zati honetan Scheduler-aren politika ezberdinak inplementatzea eskatzen da. Horretarako, schedulerraren datu egituretan aldaketak egingo dira politika ezberdin hauek onartzeko.

2. Sarrera

Proiektu honetan, 4 politika inplementatu dira. Horretarako era optimoan politika hauek inplementatzen duten oinarrizko 2 datu egitura ezberdin sortu dira. Gainera Quantum bidezko kanporaketa onartzeko sistema inplementatu da.

Beste aldetik, politiken inplementazioa era orokorrenean egiteko esfortzua egin da etorkizunean nahi diren politikak gehitzeko erraztasuna izateko. Hau lortzeko, scheduler-aren PCB kodeaketaren interfaze antzeko bat sortzea lortu da. Ondorioz, politika berri bat gehitu nahi izatekotan, interfaze hau inplementatzen duen programa bat soilik sortu beharko da, aldaketa konplexurik egin behar izan gabe.

3. Inplementatutako politikak

Errealitatean, 2 politika nagusi inplementatu dira. Hau horrela izanda ere, quantum kanporaketa gaitu eta ezgaitu daitekeenez, 2 politika hauetatik, beste bi sortzen dira. Oinarrizko politikak hurrengoak dira:

- **FIFO politika:** Lehen sartutako prozesua da lehen exekutatuko dena. Prozesua amaitu arte exekutatuko da, hau da, ez da kanporatuko.
- **Round Robin Priority Queue:** Prozesuak, hauen prioritateen arabera exekutatuak izango dira. Prioritate handiagoa izateak, lehenago exekutatuko direla adieraziko du. Prioritate berdineko prozesuek, FIFO politika erabiliko dute. Prozesuek 2 egoera izango dute ilaran daudenean: "aktibo" eta "expired" egoerak. Prozesu bat berriz listara gehitzerako unean, "expired" egoerarekin gehituko da. Prozesu hauek, "expired" egoera dutenak, ez dira berriro exekutatzeko hautagaiak izango "ktibo" egoeran dauden prozesu guztiak agortu arte. "Aktibo" egoera duen prozesurik ez dagoenean, "expired" egoeran zeuden prozesu guztiek "ktibo" egoerara aldatuko dute hauek berriro exekutatzeko hautagaiak bilakatuz.

Aurretik aipatu bezala, quantum bidezko kanporaketa era modularrean aktibatu ala desaktibatu daitekeenez, aurreko 2 politikentzako inplementatutako datu egiturak birziklatuz, hurrengo 2 politikak ere erabili ahal izango dira:

- **FIFO politika Round Robin-ekin**: FIFO politika bezala baina, kasu honetan, prozesuek quantum finko bat izango dute. Hau agortzerakoan kanporatuak eta berriz FIFO listara sartuko dira.
- **State Priority Queue**: Kasu honetan, prozesuak haien prioritatearen arabera ordenatu eta exekutatuko dira hauek amaitu arte. Dena den, ez da Priority Queue normal bat izango. Prozesuak 2 egoera izango dute: "Aktibo" eta "Agortuta". Aktibo dauden prozesuak izango dira exekutatuko direnak. "Aktibo" egoeran dauden prozesuak agortzerako orduan, "Agortuta" egoeran dauden prozesu guztiak "Aktibo" egoerara mugituko dira. Une honetan bakarrik "Agortuta" zeuden prozesuak exekutatzeko gaitazuna izango dute. Orduan, prozesu bat gehitzerako orduan, prioritatea dela delakoa, "Agortua" egoera jasoko du eta "Aktibo" dauden prozesuak amaitu arte itxaron beharko du. Hau "Aktibo" egoeran dauden prioritate baxuagoko prozesuak lehenego exekutatzea ahalbidetuko du, politika originalaren funtzionamenduarekin kontrajarriz. Hala ere, kasuren batean politika egoki bat izan daiteke.

Aurreko politikak identifikatzeko, bakoitzari aurreprozesagailuaren konstante bat esleitzea erabaki da. Horretarako hurrengo aurreprozesagaitu aldagaiak sortu dira, bat politika bakoitzeko:

- **State Priority Queue:** "`MY_SCHED_PRIO_FIFO`" konstantea "1" balioa izango duena
- **Round Robin Priority Queue:** "`MY_SCHED_PRIO_RR`" konstantea "2" balioa izango duena
- **FIFO politika:** "`MY_SCHED_FIFO`" konstantea "3" balioa izango duena
- **FIFO politika Round Robin-ekin:** "`MY_SCHED_RR`" konstantea "4" balioa izango duena

4. Soluzioaren diseinua

Proiektuaren schedulerra inplementatzeko, scheduler interfaze bat eta interfaze hau inplementatzen duten 2 politika ezberdinen funtzionalitatea zein datu egiturak eraiki dira. Gainera quantum bidezko kanporaketa onartzeko sistema ere inplementatu da.

4.1. Scheduler-aren interfazearen diseinua

Scheduler-aren politika bakoitzaren funtzionamendua eta prozesuen kudeaketaren konplexutasuna abstraitzeko interfatze bat sortzea erabaki da. Interfaze hau 2 zati nagusi izango ditu: Scheduler interfazea beraren egitura eta politika ezpezifiko baterako interfazearen inplementazioa.

Errealitatean, scheduler interfazearen bitartez, politika bakoitza inplementatzeko beharrezkoa den prozesuen PCB objektuen kudeaketa soilik burutzen da. Kudeaketa hau burutzeko, interfazeak datu egitura espezifiko bat diseinatzea eta honen gainean funtzio mugatu bat inplementatzea ahalbidetzen du. Hau da, kontextu aldaketan prozesuen hurrengo elementua proposatzeaz, hurrengo prozesuen proposamen ordena kontrolatzea

eta prozesu berrien edo kanporatuak eta bertxertatuak izan diren prozesuen ordena zehazteaz arduratuko da.

Kanporaketa prozesuan parte hartzen duten beste sistemak, quantum bidezko kanporaketa sistema adibidez, scheduler-aren beste zati bateko inplementazioaren erantzunkizuna izango da. Ondorioz, politika bat inplementatzeko, errealitatean, scheduler interfazearekin bat, scheduler-aren beste zati batzuk ere manipulatu behar izango dira. Izan ere, manipulazio hauek, interfazearen ondorioz lortutako abstrakzioari esker, nahiko gainazalezkoak izango dira.

Hortaz, sistema honen helburua, politiken inplementazio konplexuan abstrakzio bat sortzea da, inplementazio konplexuen kudeaketa sinplifikatzeko eta etorkizuneko politiken inplementazioak eta hauen integrazioa errazteko.

4.1.1. Scheduler interfazea

Scheduler interfazea, datu egitura zehatz bat eta funtzio espezifiko batzuk gordetzen dituen datu egitura bat baino ez da. Interfaze datu egitura honekin, hau kontrolatzeko funtzio laguntzaile multzo bat ere integratzen du. Gordetzen duen datu egitura hauek zein funtzioak, exekuzio momentuan era dinamikoan konfiguratu daitezke. Era honetan, interfaze objektuaren portaera alda daiteke.

4.1.1.1. Scheduler interfazearen datu egitura: "s_i_sched"

```
struct s_i_sched { // Struct_Interface_Scheduler
    void * sched_data;
    void    (* insert)  (void * sched, pcb_t * new_pcb);
    void    (* init)    (void * sched);
    pcb_t *  (* peek)    (void * sched);
    pcb_t * (* out)     (void * sched);
    char * (* print)     (void * sched);
};
typedef  struct s_i_sched  s_i_sched;
```

Scheduler interfazearen struct-a

Ikusi daitekeenez, "`s_i_sched`" struct-ak , hau da, scheduler-aren interfaze struct-ak, 6 aldagai ditu. Hauetako azkenengo 5-ak inplementatuak izan beharko diren funtzioen helbideak gordeko dituzte. Bestetik, "`sched_data`" aldagaiak, funtzioen inplementazio horretarako eta prozesuak behar bezala kudeatzeko beharrezkoa izango den datu egitura zehaztea ahalbidetuko du. Funtzio hauek, "scheduler-funtzioak" izenarekin ere erreferentziatuak izango direnak, hurrengoak izango dira eta bakoitzaren erreferentzia dagokion "`s_i_sched`" datu egituraren aldagaian gordeko dira:

- **insert**: Prozesu bat scheduler-aren datu egituran behar bezala gehituko du.
- **init**: Scheduler interfazearen inplementazioa hasieratuko du.
- **peek**: Hurrengo prozesua zein den behar bezala kalkulatu ostean, hau bueltatuko du sheduler-aren datu egituratik kendu gabe.
- **out:** Hurrengo prozesua kalkulatu, bueltatu eta scheduler-aren datu egituratik kanporatuko du.

- **print:** Scheduler-aren egoera inprimatuko du. Funtzio hau, scheduler-aren inplementazio bakoitzean erabilitako datu egitura ezberdinen informazioa kudeatu eta bistaratu ahal izateko sortu da.

4.1.1.2. Scheduler interfazearen funtzio laguntzaileak:

Beste aldetik, interfazeko scheduler-funtzioak erabiltzeko 5 funtzio sortu dira, bat scheduler-funtzio bakoitzeko. Funtzio hauek, oso sinpleak dira eta scheduler-funtzioak era egokian eta parametro egokiekin deitzen direla zihurtatzeko sortu dira. Funtzioak hurrengoak dira:

- `void i_sched_init(s_i_sched * sched)`: **"init"** funtzioa deitzen du
- `pcb_t * i_sched_peek(s_i_sched * sched)`: **"peek"** funtzioa deitzen du
- `pcb_t * i_sched_out(s_i_sched * sched)`: **"out"** funtzioa deitzen du
- `void i_sched_in(s_i_sched * sched, pcb_t * new_pcb)`: **"in"** funtzioa deitzen du
- `char * i_sched_print(s_i_sched * sched)`: **"print"** funtzioa deitzen du

Funtzio hauekin, interfazean kargatutako funtzioak era egokian exekutatzea lortuko da, interfazean kargatutako funtzioak (`init`, `peek`, `out`...) eta dagokien datu egitura ("`sched_data`") era egokian kudeatuz. Adibidez, "`i_sched_in`" funtzioaren inplementazioa hurrengoa izango da:

```
void i_sched_in(s_i_sched * sched, pcb_t * new_pcb){
    sched->insert(sched->sched_data, new_pcb);
};
```

"`i_sched_in`" funtzioaren inplementazioa

Bestetik, interfazearen inplementazioa errazten duen funtzio bat ere sortu da. Funtzio honen bitartez, "`s_i_sched`" struct-aren balioak era egokian hasieratzea ahalbidetzen da. Funtzio hau "`i_schedImplement`" izena du eta dokumentu honetan tratatuko ez den sinadura luze bat du (Hau ez du garrantzi askorik. Nahiko erraza da ulertzeko kodea bakarrik ikuskatuz). Funtzioak aurretik aipatutako datu egituraren balio bakoitzeko, hau hasieratuko duen parametro bat du. Hau interfazearen inplementazio berri bat burutu nahi duten programentzako erabilgarria izango da.

Azkenik, "`i_schedConfigure`" funtzioaren bitartez, existizen diren sheduler interfazearen inplementazio guztiak kudeatuko dira. Funtzioak, parametro baten bitartez, erabili nahi den inplementazioa hautatzea eta uneko interfaze objetuari hau aplikatzea (ala inplementatzea) ahalbidetuko du. Hau, erabilgarria izango da interfazearen inplementazioa funtzio dei bakar batez burutzeko.

Funtzio hauek biltzen dituzten fitxategiak hurrengoak dira:

- **"/include/mykernel/scheduler/scheduler_data_interface.h"**: Definizioa
- **"/src/scheduler/scheduler_data_interface.c"**: Inplementazioa

4.1.2. Scheduler interfazearen inplementazioa politika zehatz baterako

Aurreko atalean diseinatutako interfazearen inplementazio bat sortzeko 3 elementu nagusi sortu behar izango dira:

a) inplementazioa bera.
b) inplementazioa eta schaduler interfazearen funtzioak bateratzen dituzten aptadore batzuk.
c) "`i_schedImplement`" interfaze inplementazio funtzioa deitzeitzen duen eta beharrezko datu egiturak sortzen dituen funtzio bat.

Inplementazioa nahi den moduan egin daiteke beharrezkoak diren funtzioak zein datu egiturak inplementatuz. Orokorrean, politika bakoitzeko oinarrizko datu egitura bat eta scheduler-funtzioak inplementatzen dituzten prozedurak inplementatuko dira.

Zezhazki, datu egituraren inguruan, hau bakarra izan beharko da scheduler interfazeko datu egituran ("`s_i_sched`") hau gordetzeko aldagai bakarra baitago: "`sched_data`". Hala ere, datu egitura hau beste hainbat datu egitura dituen struct bat izan daiteke. Inplementazioak, aurretik aipatu bezala, beharrezkoak diren datu egiturak zein funtzioak sortzea ahalbidetuko du, eta elementu hauen funtzionamendua interfazea erabiltzen duten elementuentzako abstraituko du. Era honetan inplementazio konplexu bat erabilera erraza duen inplementatzio sinplifikatu bat bezala kudeatu ahal izango da.

Adaptatzaileen funtzioak hurrengoak izango dira. Funtzio bat aurretik aipatutako scheduler-funtzio bakoitzeko:

- `void {politika_izena}_ISchedAdpt_in(void * sched, pcb_t * new_pcb)`
- `void {politika_izena}_ISchedAdpt_init(void * sched_data);`
- `pcb_t * {politika_izena}_ISchedAdpt_peek(void * sched_data);`
- `pcb_t * {politika_izena}_ISchedAdpt_out(void * sched_data);`
- `char * {politika_izena}_ISchedAdpt_print(void * sched_data);`

Funtzioan agertutako `{politika_izena}` balioak adaptadore funtzio multzoa dagokion inplementazio sortarekin lotzeko eraliliko da. Hau da, izenak inplementatutako politikaren izenarekin lotura izango du. Funtzio bakoitzak, dagokion sheduler-funtzioaren inplementazio funtzioa deituko du parametro egokiak erabiliz. Parametro hauek, inplementazioari espezifikoa den datu egitura kudeatuko du beharrezkoak diren casting-ak burutuz. Adibidez, "`void {politika_izena}_ISchedAdpt_in(void * sched, pcb_t * new_pcb)`" adapter funtzioak "`void {politika_izena}_in(void * sched_specific_data_structure, pcb_t * new_pcb)`" inplementazio funtzioa deituko du:

```
void rq_PRIO_FIFO_ISchedAdpt_in(void * sched_data, pcb_t * new_pcb){
    rq_PRIO_FIFO_in( (rt_runqueue * ) sched_data, new_pcb);
};
```

"rq_PRIO_FIFO" edo Round Robin Priority Queue politikaren adapter funtzioaren eta inplementazioaren arteko erlazioa

Ondoren "**{*politika_izena*}_ISchedImplement(s_i_sched * isched)**" funtzioa sortuko da politika inplementazio bakoitzeko. Funtzio honen bitartez, "**s_i_sched**" interfazea aurretik aipatutako politika horien adaptatzaileekin hasieratzea lortuko da.

4.1.2.1. Round Robin Priority Queue politikaren integrazioa interfazearekin:

Round Robin Priority Queue politika inplementatzeko, aurretik aipatutako 5 adaptatzaile funtzioak sortu dira:

- `void rq_PRIO_FIFO_ISchedAdpt_in(void * sched, pcb_t * new_pcb):`
  - ↳ "`rq_PRIO_FIFO_in`" inplementazio funtzioa adaptatzen du
- `void rq_PRIO_FIFO_ISchedAdpt_init(void * sched_data):`
  - ↳ "`rq_PRIO_FIFO_init`" inplementazio funtzioa adaptatzen du
- `pcb_t * rq_PRIO_FIFO_ISchedAdpt_peek(void * sched_data):`
  - ↳ "`rq_PRIO_FIFO_peek`" inplementazio funtzioa adaptatzen du
- `pcb_t * rq_PRIO_FIFO_ISchedAdpt_out(void * sched_data):`
  - ↳ "`rq_PRIO_FIFO_out`" inplementazio funtzioa adaptatzen du
- `char * rq_PRIO_FIFO_ISchedAdpt_print(void * sched_data):`
  - ↳ "`rq_PRIO_FIFO_print`" inplementazio funtzioa adaptatzen du

Funtzio hauek, Round Robin Priority Queue politikaren inplementazio funtzioak interfazearekin bateratzeko erabili dira.

Azkenik "`rq_PRIO_FIFO_ISchedImplement`" funtzioa sortu da. Funtzio honek, aurretik aipatutako adapter-ak eta kasu honetarako egokia den datu egitura sheduler interfaze "s_i_sched" interfazean kargatzeko erabiliko da. Horretarako, interfaze objetuak eskaintzen duen "i_schesImplement" funtzioa erabiliko du. Hurrengo irudian, funtzio honen inplementazioa ikusi daiteke.

```
void rq_PRIO_FIFO_ISchedImplement(s_i_sched * isched){
    rt_runqueue * sched_data = (rt_runqueue*) malloc(sizeof(rt_runqueue));

    i_schedImplement(
        isched,
        sched_data,
        &rq_PRIO_FIFO_ISchedAdpt_in,
        &rq_PRIO_FIFO_ISchedAdpt_init,
        &rq_PRIO_FIFO_ISchedAdpt_peek,
        &rq_PRIO_FIFO_ISchedAdpt_out,
        &rq_PRIO_FIFO_ISchedAdpt_print
        );
};
```

4.1.2.2. FIFO politikaren integrazioa interfazearekin:

FIFO politika inplementatzeko, Round Robin Priority Queue politikarako sortu diren adapter zein inplementazio funtzio kopuru berdinak sortu dira. Kasu honetan, logikoa denez, sortutako adaptatzaileak FIFO politikaren inplementazio espezifikoa bateratzeko sortu dira.

Funtzio ezberdinak izan arren, aurretik aipatutako Round Robin Priority Queue inplementazioaren pausu berdinak jarraitzen dira. Izen ere, inplementazio kontzeptua eta prozedura hia berdina da.

Ondorioz, ez dira FIFO politikaren integrazioaren inguruan analisirik egingo, prozedura berdina izen desberdinekin izango litzatekeelako informazio garrantzitzu gehigarririk ez eskainiz.

4.1.3. Scheduler interfazearen integrazioa scheduler-arekin

Interfazea jada sortuta zegoen scheduler-arekin bateratzeko hurrengo aldaketak burutu dira. Lehenik schedulerra kontrolatzen duen datu egitura nagusian "`s_i_sched`" motako "**`scheduler_queue_iface_object`**" izena duen scheduler interfaze aldagai bat gehitu da. Era honetan, aurretik aipatutako interfazearen funtzioak atzigarriak izango dira scheduler-etik. Hau da, "**`i_sched_init`**", "**`i_sched_peek`**", "**`i_sched_out`**", "**`i_sched_in`**" eta "**`i_sched_print`**" funtzioak.

Interfaze objektu hau, inplementatuak dituen funtzioen bitartez, PCB-ak gordetzeko eta kudeatzeko erabiliko da. Honen bitartez, scheduler-aren funtzionamendua sinplifikatzea ahalbidtuko du, prozesuen kudeaketa scheduler-aren aldetik erraztuz eta erantzunkizun hau scheduler-interfazeari utziz.

4.1.4. Scheduler interfazearen konfigurazioa

Aurretik komentatu den bezala, scheduler interfazea konfiguratzen eta hasieratzen duen 2 funtzio ezberdin daude. Atal honetan, sheduler-aren konfigurazioan, hau da, "**`i_schedConfigure`**" funtzioan, bakarrik zentratuko gara.

Funtzio honek bakarrik 2 parametro jasoko ditu: Scheduler interfaze objektua eta aplikatu nahi den politika. Aplikatu nahi den politika zehazteko zenbaki bat erabiliko da. Zenbaki hau 3. atalean zehaztutako politikaren konstanteetako bat izan beharko da. Konstante hauetako baten balioa zehaztekotan, funtzioak politika horri dagokion interfaze inplementazioa burutuko du. Kasu honetan, funtzioak hurrengo prozedura burutuko du:

- "`MY_SCHED_PRIO_FIFO`" edo "`MY_SCHED_PRIO_RR`" balioak eskaintzekotan, "Round Robin Priority Queue" politika inplementatzen duen eta jada 4.1.2.1. atalean tratatu den "`rq_PRIO_FIFO_ISchedImplement`" funtzioa deituko da.
- Bestetik, "`MY_SCHED_FIFO`" edo "`MY_SCHED_RR`" balioak eskaintzekotan "FIFO" politika inplementatzen duen "`rq_SIMPLE_FIFO_ISchedImplement`" funtzioa deituko da (4.1.2.2. atalean tratatu da).

Gainera, behin interfazean politikarekin bat datozen funtzioak kargatu direla, "**`i_schedConfigure`**" funtzioak interfazean kargatutako datu egituren hasieraketa burutuko

du "`i_sched_init`" funtzioaren bitartez. Era honetan, konfigurazioa behin eginda scheduler interfazea guztiz erabilgarria izango da.

Konfigurazioa egiten duen funtzio hau, sheduler-aren "`sched_init`" funtziotik deituko da.

4.2. Scheduler berriaren diseinua

Shedulerraren inplementazioa 2 fitxategietan definitu da:

- **"/include/mykernel/scheduler_s.h"**: Fitxategi honetan beharrezkoak diren funtzioen sinadura eta deskribapena zehazten da. Gainera, erabiliak izango diren datu egiturak ere definitzen dira.
- **"/src/scheduler/scheduler.c"**: Aurreko fitxategian zehaztutako funtzioen inplementazioa.

scheduler_s.h

Aurretik aipatu den bezala, fitxategi honetan schedulerraren funtzioak zein datu egiturak defintzen definitzen dira. Definitzen diren datu egituren aldetik hauek hurrengoak dira:

- `struct sched_cpuThreadControl`**:** Datu egitura honen bitartez, CPU hari bakoitzaren egoera kontrolatzen da. Horretarako hurrengo elementuak biltzen ditu:
    - `struct core_hari_s * core_hari_s:` Kontrolatuko den CPU hariaren erreferentzia.
    - `pcb_t * current_running_process:` CPU harian, une horretan exekutatzen hari den prozesuaren PCB-a.
    - `int cur_process_quantum:` Une honetan exekutatzen hari den prozesuaren quantum-a. Balio honek, prozesuari geratzen zaion quantum denbora adieraziko du.
    - `int current_cpu_SchedStatus:` CPU hariaren egoera adierazten duen aldagaia. Null prozesua exekutatzen ari den ala ez adierazteko erabiliko da gehienbat.

- `struct sched_egitura`**:** Scheduler-aren oinarrizko datu egitura. Honen bitartez, scheduler-aren funtzionalitate guztia eman daiteke. Hurrengo elementuak gordeko ditu:
    - `pthread_mutex_t * sched_mutex_list:` Prozesuen lista era konkurrentean ez atzitzeko mutex-a. Prozesuen interfazeko PCB-ak gordetzen dituzten datu egituren integritatearen babes bezala jokatzen du.
    - `int current_politika`**:** Scheduler-ean kargatutako politikaren kodea gordetzen duen elementua.

    - `s_i_sched scheduler_queue_iface_object:` Scheduler interfaze objektua. Bertan PCB-ak manipulatzeko funtzioak eta datu egiturak eurkituko dira.
    - `pcb_t null_process:` Null process-aren PCB-a. Prozesu baliagarririk ez dagoenean CPU harían kargatuko den prozesua.
    - `clk_timer_s * timer:` Proiektuaren lehenengo zatian diseinatutako timer objektua. Quantum dekrementazioaren indizea lortzeko erabiliko da. Timer hau, scheduler-aren akribazioa kudetuko duen timer-a izan beharko da.
    - `struct sched_cpuThreadControl ** coreControl_array:` Core hari guztien egoerak kontrolatzen dituzten objektuen array-a. Bertan exekutatzen ari den prozesuaren PCB-a, honek oraindik kontsumitu gabe duen quantuma eta core hariaren erreferentzia aurkitu daitezke.
    - `int coreControl_arr_len:` "`coreControl_array`" array-aren luzeera.

Bestetik, datu egitura hauekin bat datozen funtzioak hurrengoak dira:

- `void sched_init:` Scheduler struct-a hasieratzen eta konfiguratzen duen funtzioa. CPU guztietan prozesu nulua kargatzen du. Gainera scheduler-aren politikarentzako beharrezkoak diren hasieraketak egiten ditu, hala nola, sheduler interfazearen inplementazio egoki bat kargatu.
- `int sched_cpuThreadControl_init:` Schedulerrak CPU-ak kontrolatzeko erabiltzen dituen datu egiturak hasieratzen ditu. Scheduler-aren hasieraketan lagungarria den funtzio bat da.
- `void sched_execute(sched_basic_t * sched):` CPU hari guztietarako schedulerra aktibatzeko saiakera egiten du. Hauetan exekutatzen den prozesuaren egoeraren arabera, schedulerra aktibatzea eta prozesu berri baten exekuzioa programatzea erabakiko du.
- `void sched_Scheduler_Dispatcher:` CPU hari bat helburu izanda, honetan exekutatzen ari den prozesua kudeatu eta beste prozesu batekin ordezkatzen du. Horretarako, beharrezkoa izatekotan, kontextu aldaketa burutzen duen funtzioak deituko ditu eta kanporatua izan den prozesua kudeatuko du, hau berriz prozesuen listan gehituz ala prozesua behinbetiko erailduz. Prozesu berririk exekuzioaren zain ez egotekotan, null prozesua kargatuko du ala jada exekutatzen ari den prozesua exekuzioan mantenduko du.
- `pcb_t * __sched_GetNextFromProcessQueue(sched_basic_t * sched):` Hurrengo prozesuaren hautaketa errazten duen funtzioa scheduler interfazearen datu egituren integritatea mutex baten bitartez babestuz.
- `pcb_t * __sched_getNextProcessAndCleanQueue(sched_basic_t * sched, struct sched_cpuThreadControl * thread_control):` Hurrengo prozesu egokia lortzen duen funtzioa. Hautaketa hau, baliagarria den lehenego prozesua aukeratuz eta bilaketa prozeduran baliagarriak ez diren prozesuak erailduz burutuko du.

Funtzio hau "`__sched_GetNextFromProcessQueue`" deituko du hurrengo PCB-a era seguruan jasotzeko.

- `void sched_AddToProcessQueue(sched_basic_t * sched, pcb_t * new_pcb)`: Scheduler interfazearen datu egituran prozesu baten txertaketa errazten duen funtzioa. Hau scheduler interfazearen datu egituren integritatea babestuz burutzen du.
- `void disp_Dispatch(pcb_t * old_pcb, pcb_t * new_pcb, struct core_hari_s * cpu_core_hari)`: CPU hari konkretu batean prozesuen arteko kontextu aldaketa burutzen duen funtzioa. Hurrengo funtzio lagungarriak deitzen ditu:
    - `void __disp_load`: CPU hari konkretu batean prozesu bat kargatzen du beharrezkoak diren erregistroen balioak berridatziz eta hauek konfiguratuz.
    - `void __disp_unload`: CPU hari konkretu batean kargatuta dagoen prozesu bat kanporatzen du. Beharrezkoak diren erregistroen balioak dagokion prozesuaren PCB-an gordetzen ditu.
- `void disp_Terminate(struct core_hari_s * cpu_core_hari,pcb_t * pcb)`: Prozesu baten exekuzioa amaitzerako orduan, honek memorian erreserbatutako espazioa eta dagokion PCB egiturak okupatutako memoria askatzen duen funtzioa.

scheduler.c

scheduler_h.s fitxetegian agertzen diren funtzioak inplementatzen dira.

Oinarrizko funtzionamendua:

Era periodikoan exekutatuko den funtzioa "`sched_execute`" funtzioa izango da. Aurretik azaldu denez, honen bitartez, CPU hari bakoitzeko hauetan exekutatzen ari diren prozesuen kanporaktea baloratuko da. Balorazio honetan, prozesuaren egoera eta, politikak quantum bidezko kanporaketa onartzekotan, honi geratzen zaion quantuma kontuan izango da.

Balorazio prozedurak prozesua kanporatua izan behar dela zehazten badu, orduan, CPU hari horretarako "`sched_Scheduler_Dispatcher`" funtzioa exekutatuko da.

Funtzio honen bitartez CPU hari batean exekutatzen hari den prozesua kanporatzen saiatzen da beste prozesu berri batengatik ordezkatuz. Sistemak "`__sched_getNextProcessAndCleanQueue`" funtzioa deituko du hurrengo prozesu baliagarria lortzeko. Hurrengo kasuak aurkitu daitezke metodoaren funtzionamendua eraldatu dezaketenak:

- Hurrengo prozesu baliagarririk baldin ez badago orduan, hurrengo frogaketa gehigarriak egingo dira erabaki logiko bat hautatzeko:
    - Uneko prozesua amaitu bada, orduan, prozesu horrek okupatzen duen memoria "`disp_Terminate`" funtzioaren bitartez askatu eta Null prozesua CPU harian kargatuko du.
    - Uneko prozesua amaitu ez bada, orduan exekuzioan dagoen prozesua CPU harían mantentzen du eta haren quantum-a berrezartzen du.
- Exekuzioaren zain dagoen prozesu bat scheduler-aren prozesuen datu egituran egotekotan, orduan jada exekuzioan dagoen prozesua kanporatu eta prozesu berria

kargatuko du "`disp_Dispatch`" funtzioa erabiliz . Gainera kanporatu den prozesuaren egoerak honen exekuzioa amaitu dela adierazten badu, orduan "`disp_Terminate`" funtzioaren bitartez honek okupatzen duen memoria askatuko du. Bestela, prozesua oraindik amaitu ez bada, hau scheduler-aren datu egituran berriz gehituko du "`sched_AddToProcessQueue`" funtzioa erabiliz.

4.3. Quantum bidezko kanporaketa

Quantum bidezko kanporaketa burutzeko 2 elementu nagusi gehitu dira. Lehenengoa, exekutatzen ari den prozesuaren quantum-a gordetzeko, "scheduler_s.h" fitxategian definitzen den "`sched_cpuThreadControl`" datu egituran "**`cur_process_quantum`**" aldagaia gehitu da. Bertan, prozesuari geratzen zaion quantum-a zehazteko da.

Scheduler-a aktibatzerako unean ("`sched_execute`" funtzioa deitzerako unean), honen frekuentzia aktibazioaren arabera (**`sched_egitura::timer`** aldagaieko struct-ean zehaztuta dagoena), unean exekuzioan dauden prozesuen quantum-ak murriztuko ditu. Ondoren, exekuzioan dauden prozesuen quantum-ak aztertu era hauetako bat agortuta egotekotan, CPU hari horretan exekutatzen ari den prozesurako, scheduler-aren prozesu kanporaketa mekanismoa exekutatzen du. Hau da, "`sched_Scheduler_Dispatcher`" funtzioa deituko da CPU hari horretarako.

Quantum bidezko kanporaketaren aktibazioa politikaren arabera gaitu ala ez-gaitzen da automatikoki. Kanporaketa metodo hau gaitzen duten politikak hurrengoak dira:

- Round Robin FIFO politika
- Round Robin Priority Queue politika

Beste politikek, exekuzioan dauden prozesuak amaitu arte ez dituzte hauek kanporatuko.

Sistemaren oinarrizko diseinu honetan, quantum-en bidezko kanporaketa aktibatzerako unean, exekutagarriak diren prozesuei beti quantum finko eta berdin bat esleitzen zaie. Etorkizunean, quantum honen balioa eta quantum-arekin zerikusirik duten beste aldagai batzuk (aurreko quantuma, CPU ziklo erabilpen koupurua, …) prozesuaren PCB-an gordetzea egokia izango litzateke. Era honetan, quantum-a dinamikoki kalkulatua izan ahalko litzateke scheduler interfazearen edo beste modulu baten bitartez. Honek CSF (Completely Fair Scheduler) eta horrelako politiken inplementazioa erraztuko luke.

4.4. Politiken diseinu eta inplementazioa

Aurretik aipatutako politikak inplementatzeko datu egitura eta, hauen gaineran, funtzionalitate ugari sortu behar izan dira. Zehazki oinarrizko politika bakoitzeko, datu egitura bat eta hau kudeatzen duten funtzio batzuk sortu dira, azken hauek sheduler interfazearekin bateragarriak izateko diseinua kontuan izanda. Hurrengo ataletan, oinarrizko politikak eta hauen inplementazioaren xehetasunak azalduko dira:

4.4.1. FIFO politika

Politika hau inplementatzeko hurrengo fitxategiak erabili dira:

- **"/include/mykernel/scheduler/schedp_simple_fifo.h"**: Futzioen interfazea zehazteko
- **"/src/scheduler/sched_simple_fifo.c"**: "schedp_simple_fifo.h" fitxategian agertzen diren funtzioen inplementazioa

Bestetik, aurreko praktikan zehaztutako linked-list datu egitura erabili da. Datu egitura hauek hurrengo fitxategietan aurkitu daitezke.

- **"/include/data_structures/lnklist_s.h"**: Linked-list-aren datu egiturak eta funtzioen interfazea definitzen da
- **"/src/datu_egiturak/lnklist_LFRL.c"**: Aurreko fitxategian zehaztutako funtzioen inplementazioa.

Errealitatean, hau da aurreko praktikan inplementatua izan zen politika. Proiektuaren zati honetan, inplementazioa aurretik aipatutako interfaze egiturara moldatzeko aldaketak baino ez dira egin. Ondorioz, ez dira emango politika honen inplementazioaren inguruko xehetasun gehigarririk.

Dena den, linked-list datu egitura zuzenean erabili beharrean, hau hurrengo atalean definituko den "`rt_simple_runqueue`" struct sinple baten barnean bildu da "runqueue" edo exekuzio listaren kontzeptuarekin bat egiteko eta etorkizunean gehitu ahal izango diren funtzionalitate berrien inplementazioa sustatzeko eta errazteko.

4.4.2. Round Robin Priority Queue politika

Politika hau inplementatzeko hurrengo fitxategiak erabili dira:

- **"/include/mykernel/scheduler/schedp_simple_fifo.h"**: Futzioen interfazea zehazteko
- **"/src/scheduler/sched_simple_fifo.c"**: "schedp_simple_fifo.h" fitxategian agertzen diren funtzioen inplementazioa burutzen da

Bestetik, politika inplementatzeko "runqueue" datu egitura sortu da. Datu egitura honen definizioa hurrengo 2 fitxategietan banatu da:

- **"/include/data_structures/runqueues_s.h"**: Datu egituren eta hauek hasieratzeko oinarrizko funtzioen interfaze definizioa egiten da.
- **"/src/datu_egiturak/runqueues.c"**: "runqueues_s.h" fitxategian agertzen diren funtzioen inplementazioa burutzen da

Azkenik, "runqueue" datu egiturak FIFO politikan inplementatutako "linked-list" datu egitura ere erabiltzen duela aipatu behar da. Gainera, prioritateak onartzeko, PCB struct-ean prozesuaren prioritatea kudeatuko duen aldagaia gehitu da.

runqueues_s.h

Fitxategi honetan hurrengo datu egiturak definitzen dira:

- `struct rt_prio_array`: Tamaina finko bateko array bat duen struct-a. Array honek lista estekatu ezberdin bat (`lnklist_LFRL`) izango du array-aren posizio bakoitzeko. Posizio bakoitzeko listan, pozizioari dagokion prioritate zehatz bateko prozesuak gordeko dira. Bestetik, array-ean une horretan dauden prozesu guztien kontaketa eramaten duen aldagai bat izango du.
- `struct rt_runqueue`: Politika honetan prozesuak kudeatzeko erabiliko den datu egitura nagusia da. Bi "`rt_prio_array`" ditu. Hauetako bat prozesu aktiboak eta besteak expiratutako prozesuak gordeko ditu. Elementuak beti, prozesu expiratuen listan gehituko dira eta prozesu aktiboen listatik kanporatuak izango dira. Prozesu aktiboen lista hutsik bilakatzen den unean, prozesu expiratuen listako elementuak, prozesu aktiboen listara mugituko dira.
- `struct rt_simple_runqueue`: "`lnklist_LFRL`" datu egitura bakarra eta bertan dauden prozesuen kontakera eramaten duen aldagaia du. Datu egitura hau, "FIFO" politika inplementatzeko diseinatu da.

Bestetik, datu egitura hauek kontrolatzeko hurrengo funtzioak definitu dira:

- `void prio_array_init(rt_prio_array * pri_arr)`: "`rt_prio_array`" struct baten hasieraketa burutzen du. Array-ean dauden lista estekatu guztiak ere hasieratzen ditu.
- `void runqueue_init(rt_runqueue * runq)`: "`rt_runqueue`" objektua hasieratzen du bertan dauden expiratutako eta aktibo dauden prozesuen "`rt_prio_array`" objektuak hasieratuz.
- `void simple_runqueue_init(rt_simple_runqueue * runq)`: "`rt_simple_runqueue`" objektua hasieratzen du. Honen bitartez, struct horretan dagoen lista estekatua hasieratuko da ere.

runqueues.c

"runqueues_s.h" fitxategian definitutako funtzioak inplementatzen dira.

schedp_prio_fifo.h

Fitxategi honetan politika inplementatzen duten hurrengo funtzioak definitzen dira:

- `void rq_PRIO_FIFO_in(rt_runqueue * sched, pcb_t * new_pcb)`: Prozesu bat, expiratutako prozesuen listan ("`rt_prio_array`"-ean) gordetzen du. Hau da, haren prioritatearen arabera dagokion lista estekatuan gehituko du.
- `void rq_PRIO_FIFO_init(rt_runqueue * sched)`: Scheduler interfazearekin bateragarritasuna bermatzen duen funtzioa. Honen bitartez, parametroz pasatutako "`rt_runqueue`" motako "`sched`" objektua hasieratzen du.

- `pcb_t * rq_PRIO_FIFO_peek(rt_runqueue * sched)`: "`rq_PRIO_FIFO_out`" funtzioa bezala baina, kasu hoentan, ez da prozesua listatik erauzten. Hau da, hurrengo prozesuaren PCB bueltatzen da hau datu egituratik borratu gabe.
- `pcb_t * rq_PRIO_FIFO_out(rt_runqueue * sched)`: "`_rq_PRIO_FIFO_updateSched`" funtzioa deitu ondoren, prozesu aktiboak dituzten prioritate array-eko listetan, hurrengo prozesua topatzen saiatzen da. Array-ean dauden listak prioritate handieneko listetatik prioritate baxueneko listeetara arakatzen dira, prioritate handien dituzten prozesuak lehenik topatuz. Hutsik ez dagoen lista bat topatzen den momentuan, lista honetatik hurrengo PCB-a erauzten du. Erauzitako PCB hau izango da funtzioak bueltatuko duena.
- `char * rq_PRIO_FIFO_print(rt_runqueue * sched)`: Datu egituraren egoera bistaratzeko erabiltzen da. Datu egituran dauden prozesuak bistaratzen ditu hauen prioritatea eta expiratu edo aktibo egoera pantaiaratuz.
- `int _rq_PRIO_FIFO_updateSched(rt_runqueue * sched)`: Prozesu aktiboen datu egituran dauden prozesu kopurua 0 izatekotan, prozesu aktiboen zerrenda prozesu expiratuen zerrendagatik trukatzen du.

schedp_prio_fifo.c

"schedp_prio_fifo.h" fitxategian definitutako funtzioak inplementatzen dira.

Funtzionamendu orokorra:

Azken finean, aurretik aipatutako funtzioak zein datu egiturak erabiliz "Round Robin Priority Queue" politikaren funtzionamendu osoa lortuko da.

5. Integrazioa Main programan:

Aurretik aipatutako sistema martxan jartzea nahiko erraza da. Horretarako bakarrik hurrengo funtzio deiak aurrera eraman beharko ditugu:

- **Lehenik politika aukeratu:**
  Diseinatutako 4 politiken artean aukeratzeko hurrengo sistema erabiliko da:

```
int politika = 0;
// Politika aukeraketa
if(argc >= 2 && strspn(argv[1], "0123456789") == strlen(argv[1])) {
    sscanf(argv[1], "%zu%*c",&politika);
}
printf("Scheduler Politika:\n -");
switch(politika){
        case MY_SCHED_FIFO:
            printf("Simple FIFO\n");
        case MY_SCHED_RR:
            printf("Round Robbin\n");
        case MY_SCHED_PRIO_FIFO:
            printf("Priority FIFO\n");
        case MY_SCHED_PRIO_RR:
            printf("Priority FIFO with RR\n");
            break;
        default:
            printf("DEFAULT: Priority FIFO with RR\n");
            politika = MY_SCHED_PRIO_RR;
            break;
    }
```

Politika hautatzea ahalbidetzen duen kodea

Sistema honen bitartez programari parametro bitartez pasatutako zenbakiaren arabera politika ezberdin bat hautatuko da. Politika bakoitzari dagokion zenbakia jakiteko "3. Inplementatutako politikak" atalaren amaieran aurkitu daitezke.

- **Jarraitzu schedulerraren "`sched_execute`" funtzioa timer bati ("`clk_timer_init`" erabiliz) lotu:**

```
clk_timer_init(&tmr1_scheduler,&clkm_1,
    SCHEDULER_FREQUENCY, &sched_execute, &my_scheduler, "Sheduler");
```

Era honetan, timer-ak exekutatu ondoren, hauek konfiguratutako denboran "`sched_execute`" funtzioari deituko diote.

- **Ondoren schedulerra hasieratu "`sched_init`" funtzioa erabilitz:**

```
sched_init(&my_scheduler, &tmr1_scheduler, my_cpu_arr,CPU_KOP, politika);
```

Funtzio honi, hasieratu gabe dagoen scheduler objektuaren helbidea, honen timer-a, kudeatuko dituen CPU-ak eta aurretik kalkulatutako politika pasatuko zaizkio parametro bitartez.

- **Azkenik timer-a eta erlojuak exekutatu:**

```
if (pthread_create(&tr_tmr1_sched, NULL, clk_timer_start, (void *) &tmr1_scheduler))
    printf("Error: Timer started\n");
```

Aurreko irudiaren bitartez, scheduler-aren timer-a exekutatuko da hari berri batean.

```
printf("Clock execute\n");
if (pthread_create(&tr_clk, NULL, clk_clock_start, (void *)  &clkm_1))
    printf("Error: Clock started\n");

pthread_join(tr_clk, NULL);
```

Amaitzeko, timer guztiak kontrolatzen dituen erlojua exekutatuko da.

6. Garatutako beste hobekuntza batzuk:

Aurreko praktikarekin alderatuz, Timer eta Erlojuen API-a sinplifikatu da. Hau lortzeko "`clk_clock_init`" eta "`clk_timer_init`" funtzioak sortu dira, erlojua eta timer-ak hasieratzea errazteko hurrenez hurren. Era honetan, mutex-aren kontrolaren abstrakzioa egin da, programaren erabilpen konplexutasuna sinplifikatuta.

7. Etorkizuneko lana:

Prozesuen kanporaketa kliskagailu (disparadore) metodo ezberdinak inplementatzen dituen modulua. Era honetan, quantum eta gertaera ezberdinen bidezko kanporaketak onartuak izango litzateke.

Beste aldetik, quantum kalkulu eta honen esleipen dinamikoa ahalbidetzen duen modulua inplementatzea ondo egongo litzateke.