2018-19

Sistemes Operatius II

PRÀCTICA 5

FRANCISCO DÍAZ RUIZ NIUB: 16828405

JOSÉ BLANCO FERNANDEZ NIUB: 18088022

- **OBJECTIUS:**

  Per aquesta pràctica havíem de modificar el codi de la pràctica 4 de tal d'implementar el sistema de productors i consumidors que utilitzen un buffer comú on els productors posen dades dintre i els consumidors les extreuen. En el nostre cas s'havia de fer amb l'objectiu de que es llegís les dades dels vols pels diferents aeroports.

- **IMPLEMENTACIÓ:**

  Partint del codi de la pràctica anterior, que implementava fils per tal de omplir l'arbre amb les dades corresponents a cada node/aeroport, ara cal substituir aquesta implementació per una que utilitzi productors i consumidors. Per fer-ho hem hagut de crear diferents estructures dintre de *ficheros-csv.h* que corresponen a les diferents dades que han de utilitzar tant els productors com els consumidors.

  Per tal de poder administrar bé els productors i els consumidors, s'han hagut de crear diferents elements, aquests són:
  - Un *pthread_mutex_t* que només l'utilitzaran els productors i els consumidors.
  - Els propis productors i consumidors, sent aquests objectes *pthread_t.*
  - Les condicions per els productors i els consumidors, de tipus *pthread_cond_t.*
  - El buffer que compartiran entre ells (*buff_pc).*

  Tanmateix, hem creat diferents classes que s'encarreguen de dur a terme les accions de llegir el fitxer, omplir i buidar el buffer i, finalment, omplir l'arbre de dades. Aquestes classes són:
  - *Void *Producer(void * args);*
  - *Void *consumer(void *args);*
  - *Void *read_file(void *args);*
  - *Void *process_file(void *args);*

  I les estructures que aquestes classes utilitzaran, que son les següents:
  - *Typedef struct read_par*
  - *Typedef struct process_par*
  - *Struct buffer*
  - *Struct cell*

  Com la implementació dels productors i els consumidors s'encarreguen de fer les mateixes tasques que la implementació de fils que vam haver de realitzar en l'anterior pràctica, la funció *read_airports_data* ja no cal utilitzar-la, encara així hem decidit deixar-la implementada però no s'utilitza en cap moment. En relació amb això hem hagut de tornar a posar el comptador de temps d'execució que aquesta classe portava implementat. Nosaltres l'hem posat dintre de la funció *create_tree.*

Per la part del productors, hem implementat en la funció *void *Producer(void *args)* el seu funcionament. Aquest primer de tot agafa els arguments que entren per paràmetre i els passa a tipus *struct read_par*, després fa *pthread_mutex_lock* del mutex que només utilitzen els productors i els consumidors, seguidament mira si tots els espais per fils estan ocupats, si es així entra en espera dintre del *pthread_cond_wait(&cond_producer, &mutex_prod_consum)* fins que acabi un dels fils, després agafarà les dades de dintre del fitxer i les col·locarà dintre del buffer compartit per tal de que un consumidor agafi aquestes dades. Finalment fa un *pthread_cond_broadcast(&cond_consumer)* i després *pthread_mutex_unlock(&mutex_prod_consum).*

Llavors passaríem a la implementació dels consumidors, que agafen les dades que hi ha en el buffer i les introdueixen dintre del node corresponent en cas de ser noves o actualitza el node en cas de que ja hagin dades prèviament. Per tal de dur a terme aquestes accions, al igual que els productors, agafen les dades entrades per paràmetre i les passa a format *struct process_par* i, seguidament bloqueja el *mutex*. Al igual que en la implementació dels productors, els consumidors miren si hi ha espai lliure, i en cas negatiu esperen a dintre del *pthread_cond_wait(&cond_consumer, &mutex_prod_consum).* Una vegada ja poden passar miren si hi ha dades dintre del buffer, si es positiu, llavors passa a comprovar les dades i si son valides farà el procés de introduir-les dintre del arbre.

Finalment, a la funció *create_tree* es crea l'arbre amb tots els nodes que s'han llegit del fitxer aeroports.csv i, seguidament, inicialitzem el buffer i les cel·les d'aquest. Seguidament, s'inicialitza també els *structs read_par i process_par* i es creen els diferents fils que utilitzaran aquests amb *pthread_create*. Llavors aquest fils llegiran el fitxer de dades (els productors només) i ompliran l'arbre amb aquestes dades (els consumidors). Per acabar, es farà free de cada struct (ja que s'inicialitzen amb un malloc) i es buidarà el buffer per després fer free d'aquest.

- ## COMPLICACIONS I OBSERVACIONS:

En conclusió, no hem tingut moltes complicacions encara que ens ha costat una mica entendre el funcionament dels productors i els consumidors i com aquests no podien entrar en les parts critiques dels altres, és a dir, quan un productor estava escrivint dintre del buffer un consumidor no podia accedir a la posició del buffer on aquest productor estava escrivint o es produiria un problema. Però a part d'això no hi ha hagut moltes més complicacions.