Task Manager Application

Programozói dokumentáció

Király Bálint

Tervezési megfontolás

A program input-output rendszere egy egyszerű elv alapján funkcionál, ahogy az a main függvényből is látszik:

```
while (activeSession) {
      display(next_page);
      next_page = navigate(next_page);
}
```

Megjeleníti az adott oldal tartalmát, majd az inputnak megfelelően cselekszik. Az elgondolás hasonlít egy képfrissítéses alkalmazásra, itt is minden alkalommal újra generálja a kijelzett tartalmat. Ez a gondolat végigkíséri az egész program tervezését.

Modulok

A program 5 modulra van osztva.

main.c

Csupán a következő oldal frissítése történik itt, illetve az indításkor meghívja a `start_session()` függvényt.

A következőleg megjelenítendő oldalt egy `Page` típusba tölti mindig a `navigate()` visszatérési értéke, így frissül folyton az aktuális oldalnak megfelelően a kijelzett tartalom.

session.c

Ez a modul kezeli a session indítást és befejezést a `start_session()` és `end_session()` függvényével. A program indításakor indul egy aktív session, ezt reprezentálja az `activeSession` változó, amíg ennek az értéke igaz, addig fut a main ciklusa.
Induláskor létrejön egy `session` globális struktúra is. Ebben tárolja a program az adott session-höz tartozó, a felhasználó azonosítására való, vagy a futás közben végig elérendő változókat. Azért szükséges ennek a használata, mert a különben a különböző modulok önálló, párhuzamos munkavégzése nem lenne megvalósítható.

```
typedef struct Session {
    char user[LEN_UNAME+1];
    char password[LEN_PW+1];
    char log[200];
    Task *data;
    Task *task;
} Session;
```

`session.log`: tervezési megfontolás következményében a rendszer hiba-, és egyéb üzeneteit, a folyton törlődő konzol képe miatt ilyen formában, egy meghatározott helyről olvassa mindig ki. Ez a `session.log`.
A `session.data` az adatbázis kezdőpointerét tárolja, a `session.task` egy ideiglenes tároló, ahonnan ki és be lehet olvasni feladatokat.

A terminálban végrehajtott egyes funkciók szintaktikája különbözik operációs rendszertől függően (pl. system("cls"), vagy system ("clear")), ezért a kompatibilitás miatt a header fileban meghatározza az adott platformot.

```
#ifdef __APPLE__
#define PLATFORM 1
#else
#define PLATFORM 0
#endif
```

Ebben a modulban történik a felhasználó azonosítása. A `login_user()` függvény kikeresi az azonosítót a `users.txt` file-ból, ahol a felhasználónevek és hozzájuk tartozó jelszavak vannak tárolva. ha nem találja új felhasználóként kezeli, és a `register_user()` beírja a `users` file-ba. Ha szerepel benne, ellenőrzi a jelszó helyességét, és amennyiben helyes, belépteti a felhasználót.

`FILE *open_savefile(char *mode)`
Visszatér a megfelelő módban megnyitott save file-lal, NULL, ha sikertelen.

Ezen kívül itt található a `file_init()`. Ez létrehozza a megfelelő filet, amennyiben nem létezik még.

uinterface.c

Ez a modul felelős a user interface-ért, vagyis mindazért, amit a felhasználó lát.
A header file-ban definiáljuk a program oldalait egy felsorolt típusban:

```
typedef enum Page{
    login,
    dashboard,
    tasks,
    edit,
    create,
    logout
} Page;
```

A modul fő függvénye a `display()`. Ez törli a konzolt, majd meghívja az aktuális oldalnak kijelzéséhez megfelelő függvényt, végül kiírja a `session.log`-ot. Azért kerültek külön függvényekbe az oldalak, mert habár ugyanazt a funkciót látják el, teljesen más szerkezetűnek kell lenniük. Az egyes megjelenítőket viszont nem érdemes külön taglalni, mindegyik az oldalnak megfelelő megjelenítést valósítja meg, esetleg a `session.task` felhasználva.

navigation.c

A navigation modul hasonló a uinterface-hez abból a szempontból, hogy itt is minden alkalommal az aktuális oldaltól függő ág hajtódik végre.
A `navigate()` függvény visszatérési értéke a következőleg megjelenítendő oldal. Működése felfogható egy állapotgépként: kimenetét az előző állapota és a bemenet határozza meg. Először beolvassa a user inputot, majd minden esetben ellenőrzi, hogy az adott oldalnak megfelelően érvényes-e. Ha nem, akkor egy hibaüzenetet ír a `session.log`-ra, és ugyanoda tér vissza.
Itt történik a task különböző paramétereinek beolvasása, a bejelentkezés vezérlése, és ugyancsak az összes oldalon található opciók kiválasztása.

datahandler.c

Ez a modul végzi a program érdemi részét: olvassa, írja, és kezeli az adatbázist.

Az egyes taskokat struktúraként tárolja, melyben az összetartozó értékek szerepelnek.

```
typedef struct Date {
    int y;
    int m;
    int d;
} Date;
```

```
typedef struct Task {
    char name[LEN_T_NAME+1];
    Date due;
    char cat[LEN_T_CAT+1];
    char dscr[LEN_T_DSCR+1];
    bool done;
    struct Task *next;
} Task;
```

Az adatbázist rendezett láncolt listaként valósítja meg, ahol a listaelemek az egyes Taskokat jelölik.

Függvényei:

`Task* task_init(void)`
: Dinamikusan foglal egy Taskot, majd feltölti default értékekkel.

  Például új task létrehozása előtt a `session.task`-ot inicializálja, hogy a képernyőn üres mezők jelenjenek meg.

`void print_tasks(void)`
: Attól a `Task`-tól kezdve, amire a `session.task` mutat, kiír 3 taskot a platformnak megfelelő megjelenítésben (box-drawing karakterek nem támogatottak windowson).

`bool valid_task(Task *task)`
: Ellenőrzi, hogy a name és due mezők ki vannak-e töltve.

`Task* add_task(Task *data_start, Task *newTask)`
: `newTask`-ot `due` szerint rendezve hozzáfűzi a láncolt listához.

  Alkalmazása: új task mentése esetén a `session.task` által mutatott taskot fűzi hozzá. Betöltéskor a save fileból beolvasott taskokat menti az adatbázisba.

  Visszatérési értéke a láncolt lista első elemére mutató pointer, NULL, ha a mentés sikertelen volt.

`Task* remove_task(Task *data_start, Task *rmTask)`
: Eltávolít egy feladatot a listából, visszatér az eleje pointerrel.

`bool load_data(void)`
: Sikeres bejelentkezést követően a save file-ból betölti az adatstruktúrába az adatokat.
  Segédfüggvényei az `open_savefile()`, `load_task()`, és `insert_task()`.
  Visszatérési értéke igaz, ha a betöltés sikeres volt.

`bool save_data(void)`
: Kijelentkezéskor végigmegy a láncolt listán, kiírja és felszabadítja az elemeket.

Az üres értékekhez * karaktert ír, ezért ez a karakter tiltva van a paraméterekben.

`int compare_dates(Date d1, Date d2)`

Összehasonlít két `Date` struktúrát. Értéke -1, ha d1<d2, 0, ha d1=d2, 1, ha d1>d2.

`double percent_today(int *count_today, int *count_today_done)`

A dashboardon kijelzett mai statisztikát állítja elő.

Visszatérési értéke -1, ha nincs mai feladat, egyébként a mai kész feladatok és mai feladatok arányát adja vissza egész százalékban. A pointerként átvett változókba beírja a mai feladatokat, és a mai kész feladatokat.

`bool find_next_task(void)`

Megkeresi a legközelebbi taskot (dátum egyezés esetén a betűrendben előbb lévőt), és ráállítja a `session.task` pointert. Értéke false, ha nincs következő task.

`bool jump_seq(void)`

Amennyiben lehet, előre ugrik a láncolt listán 3-at, és ezt az új pointert írja bele `session.task`-ba. Értéke false, ha nincs már több, mint 3 elem hátra.

Környezet

A program a szabványos könyvtárakon kívül nemigen használ más forrást.
A programfile-ok mellett egy `users.txt` állomány és a felhasználókhoz tartozó `save_[username].txt` nevű file-ok szükségesek a működéshez, de amennyiben ezek nem léteznek, a program magától létrehozza azokat.