Task Manager Application

Programozói dokumentáció

Király Bálint

Tervezési megfontolás

A program 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.

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 működik a main ciklus.
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.

```
typedef struct Session {
    char user[LEN_UNAME+1];
    char password[11];
    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 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, ha nem találja új felhasználóként kezeli, és a `register_user()` beírja a `users` file-ba.

Ezen kívül két függvény található még itt: a `file_init()`, és a `dir_init()`. Ezek létrehozzák a megfelelő file-ot, vagy mappát, amennyiben nem léteznek 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,
    single,
    newTask,
    logout
} Page;
```

A modul fő függvénye a display(). Ez meghívja az aktuális oldalnak kijelzéséhez megfelelő függvényt. Azért kerültek külön függvényekbe az oldalak, mert habár ugyanazt a funkciót látják el, teljesen más szerkezetűnek kell lenniük. Minden függvényhíváskor törli az aktuális tartalmat a konzolon, ezáltal a legfrissebb értékek jelennek csak meg mindig.

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_NAME+1];
    bool done;
    struct Task *next;
} Task;
```

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

Függvényei:

`void init_task(Task *task)`
: A paraméterként kapott Taskot feltölti default értékekkel. Új task létrehozása előtt a `session.task`-ot inicializálja, hogy a képernyőn üres mezők jelenjenek meg.

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

`bool append_task(Task *data_start, Task *newTask)`
: A `navigate()` függvény beolvassa az új Task létrehozásakor bevitt paramétereket a session.task-ba, az általános Task adattárolóba.
A Task mentésekor fut le a függvény, feladata a session.task paramétereit hozzáfűzni a felhasználó save file-jához. A `data_start` a láncolt lista első elemére mutat.
Visszatérési értéke igaz, ha a mentés sikeres volt.

`bool load_data(void)`
: A save file-ból betölti az adatstruktúrába az adatokat. Két helyen van rá szükség: sikeres bejelentkezést követően, illetve új task létrehozása után.
Segédfüggvényei az `open_savefile()`, `load_task()`, és `insert_task()`.
Visszatérési értéke igaz, ha a betöltés sikeres volt.

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

`Task* load_task(FILE *savefile, bool *empty)`

Egy Task paramétereit kiolvassa a save file-ból, majd ezzel visszatér. Null, ha sikertelen, és empty értékét igazra állítja, ha nincs már task.

`Task* insert_task(Task *data_start, Task *loadTask)`

A paraméterként kapott Taskot hozzáfűzi a `data_start` elejű láncolt listához. Visszatér a lista első elemére mutató pointerrel.

`void print_tasks(void)`

Végigmegy az adatstruktúrán és kiírja az összes Task paramétereit.

Környezet

A program a szabványos könyvtárakon kívül nemigen használ más forrást, egyedül a sys könyvtárat, ami a mappa létrehozásához szükséges.
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.