

# 1 Zadanie

Skrytka pocztowa jest miejscem, w którym listonosz zostawia przesyłki zaadresowane na daną skrytkę, a klient mający klucz do skrytki może korespondencję z tej skrytki odebrać.

Zadanie polega na napisaniu systemu symulującego działanie skrytek pocztowych w pewnym urzędzie pocztowym za pomocą kolejek komunikatów i pamięci współdzielonej. System składa się z trzech programów:

```
poczta.c      zarządzającego skrytkami;
listonosz.c   roznoszącego przesyłki;
klient.c      odbierającymi przesyłki.
```

Program poczta uruchamiany jest z dwoma parametrami  $n_1$   $n_2$  — pierwszy parametr  $n_1 \geq 0$  oznacza ilość skrytek dostępnych w urzędzie, a parametr drugi  $n_2 \geq 0$  oznacza pojemność każdej skrytki. Zadaniem programu poczta jest obsługa skrytek.

Program klient uruchamiany jest z jednym parametrem  $n$ . Po uruchomieniu, proces klienta zgłasza się do serwera poczta w celu założenia skrytki otrzymuje w zamian numer skrytki, który wypisuje na standardowe wyjście. Po przeczekaniu na losowej ilości czasu proces w pętli  $n$ -razy dokonuje następujących operacji:

sprawdza czy w skrytce coś się pojawiło, jeśli tak wypisuje odczytaną ze skrytki wiadomość na standardowe wyjście. Sprawdzenie i odczytanie wiadomości trwa od 2 do 5 sekund. Jeśli nie było żadnej wiadomości wypisuje komunikat \*\*\*\*\* Brak wiadomości \*\*\*\*\* czeka losową ilość czasu. Następnie likwiduje skrytkę, odbierając ewentualną wiadomość.

Program listonosz uruchamiany jest z jednym parametrem plik, który jest nazwą pliku z zadaniami. Plik ten jest plikiem tekstowym i ma następującą strukturę: Każda linia jest postaci NR\_SKRYTKI WIADOMOSC lub jest pustą linią, która oznacza koniec partii przesyłek. Przykład pliku zadań:

```
1 wiadomosc pierwsza
2 wiadomosc druga

1 wiadomosc trzecia
5 wiadomosc czwarta
3 wiadomosc piata

6 wiadomosc szosta
```

Zakładamy, że plik z zadaniami jest zawsze poprawny.

Listonosz dostarcza przesyłki w partiach — po rozniesieniu każdej partii odpoczywa losową ilość czasu. Listonosz

1. jeśli skrytka jest zarezerwowana i pusta (odebrano wszystkie poprzednie wiadomości) listonosz zapisuje aktualną wiadomość do skrytki i wypisuje na standardowe wyjście komunikat o dostarczeniu przesyłki.
2. jeśli skrytka nie jest zarezerwowana listonosz zapomina nie dostarcza wiadomości i wypisuje komunikat, że adresat jest nieznany
3. jeśli skrytka jest zajęta (nie odebrano poprzedniej wiadomości) — listonosz wypisuje o tym komunikat i usiłuje dostarczyć daną wiadomość razem z następującą partią

Listonosz spędza na rozłożeniu wiadomości (ilość wiadomości do rozłożenia)\* $n$  sekund, gdzie  $n$  jest liczbą naturalną z przedziału  $[2,5]$ .

Uwagi:

1. istnieje tylko jeden proces poczta
2. może istnieć wiele procesów listonosz
3. istnieć wiele procesów klienta
4. proces poczta powinien przydzielać skrytki o najniższym możliwym numerze
5. w trakcie gdy listonosz wkłada wiadomości do skrytek, nikt inny nie może mieć dostępu do skrytek.

## 2 Krótki opis działania

### 2.1 Kolejki

Poczta ma kolejkę, na którą klienci i listonosze wysyłają zapytania, oraz kolejkę skrzynek wolnych. Dodatkowo poczta ma także kolejkę komunikatów odłożonych, gdzie trafiają wiadomości do ponownego wysłania, jeśli kolejka odbiorcy była pełna.

Klient i listonosz otwierają kolejkę poczty oraz tworzą własne kolejki, na które odbierają wiadomości od poczty

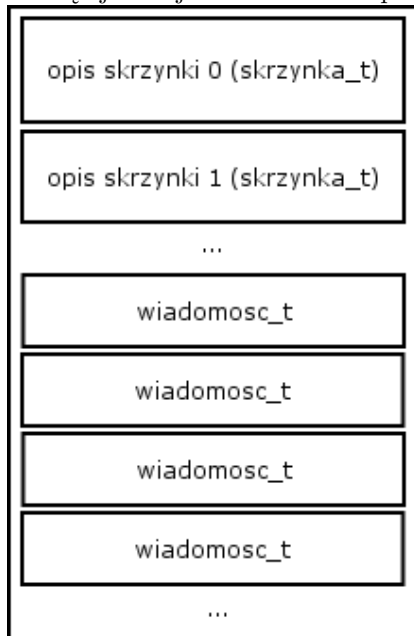
Listonosz ma jeszcze kolejkę wiadomości w partii, które ma akurat wysłać, oraz kolejkę wiadomości nie wysłanych z powodu pełnej skrzynki. Wiadomości nie wysłane wysyłane są w następnej kolejce.

### 2.2 Pamięć skrzynek

Pamięć ze skrzynkami jest współdzielona przez wszystkie procesy poczty, klientów i listonoszy. Tworzeniem jej i usuwaniem zajmuje się poczta. Poczta również na początku formatuje obszar, zerując pola właścicieli, oraz ustawiając wskaźniki do wiadomości danej skrzynki.

Ponieważ każdy proces może sobie zmapować tę pamięć w trochę innym miejscu, wskaźniki do wiadomości podają adres względny, od początku obszaru skrzynki. Każdy proces przed dostępem do wiadomości musi sobie więc powiększyć ten adres o swój adres początkowy obszaru pamięci.

Pamięć jest zajmowana w taki sposób, że najpierw jest miejsce dla skrzynek, a potem dla wiadomości.



### 2.3 Semaforey

Nie każdy proces może zawsze uzyskać dostęp do pamięci. Poczta i listonosz odwołują się po niej przede wszystkim, aby coś zapisywać, więc potrzebują wyłączości. Natomiast klienci tylko czytają, więc może ich być wielu na raz.

Stąd semaforey:

- liczący liczbę listonoszy aktualnie obecnych i mających/chcących uzyskać dostęp do pamięci
- liczący liczbę klientów

Dodatkowo jest też semafor liczby wolnych skrzynek, wykorzystywany tylko przez pocztę.

### 2.4 Uruchomienie

```
bash:~/pp_doc/so/poczta\$ ./poczta
Używanie: poczta lskrzynek pojemnosc
```

Jako argument należy podać liczbę skrzynek i ich pojemność

```
bash:~/pp_doc/so/poczta\$ ./listonosz
Używanie: listonosz plik
```

Argument wskazuje nazwę pliku z wiadomościami

```
bash:~/pp_doc/so/poczta\$ ./klient
Używanie: klient lprob
```

Liczba prób mówi ile razy sprawdzać skrzynkę

## 2.5 Zakończenie

Zamknięcie poczty możliwe jest poprzez SIGINT. Klient i listonosz zamykają się same po wykonaniu swoich zadań.

# 3 Kod źródłowy

## 3.1 struktury.h

```
#ifndef STRUKTURY
#define STRUKTURY

/* stany logiczne zwracane przez funkcje */
#define FALSE 0
#define TRUE (!FALSE)

/* stany zwracane przez aplikacje po zakonczeniu pracy */
#define APP_SUCCESS 0
#define APP_FAILED 1

/* numery semaforow */
#define SEM_LK 0 /* semafor z liczba klientow aktywnych */
#define SEM_LL 1 /* semafor z liczba listonoszy aktywnych */
#define SEM_LS 2 /* semafor z liczba wolnych skrzynek */

/* struktura komunikatow klient<->poczta, listonosz<->poczta */
enum nadawca {NADAWCA_PO CZTA=1, NADAWCA_KLIENT, NADAWCA_LISTONOSZ};
enum oper {OPER_NOWA_SKRZYNK A, OPER_USUN_SKRZYNK E, OPER_ILE_MIEJSC A, OPER_ILE_SKRZYNEK};

typedef struct zapytanie {
    long    nadawca;
    int     adr_zwrotny;
    enum oper operacja;
    int     numer;
} zapytanie_t;
#define ROZM_TRESCI (sizeof(zapytanie_t)-sizeof(long))

/* struktura skrzynki */
typedef char wiadomosc_t[128];

typedef struct skrzynka {
    char     uzywana;
    int      lwiadomosci;
    wiadomosc_t* wiadomosci;
} skrzynka_t;

/* ftok */
#define SCIEZKAPOCZTA "/tmp/poczta"
```

10

30

40

```
#endif
```

### 3.2 poczta.c

```
#include <stdio.h>
#include <sys/types.h> /* key_t */
#include <linux/ipc.h>
#include <linux/shm.h>
#include <linux/sem.h>
#include <linux/msg.h>
#include <unistd.h>
#include <errno.h>
#include <stdarg.h> /* va_list ... */
#include <time.h> /* sizeof struct tm */
#include <signal.h> /* SIGINT */
#include "struktury.h"

#define ROZM_SKRZYNEK lskrzynek*(sizeof(skrzynka_t)+(pojemnosc*sizeof(wiadomosc_t)))

int      lskrzynek=0; /* liczba skrzynek dostepnych na poczcie */
int      pojemnosc=0; /* ile wiadomosci pomiesci skrzynka */
key_t    pocztaid;    /* key dla funkcji IPC */
skrzynka_t* skrzynki; /* przestrzen wspoldzielona zawierajaca skrzynki */
int      skrzynki_wolne; /* kolejka z wolnymi skrzynkami */
int      skr_id;        /* id segmentu pamieci wspoldzielonej */
int      sem_id;        /* id semaforow IPC */
int      kolejka;       /* kolejka z zapytaniami klientow/listonoszy */

void loguj(char *format, ...) {
    va_list ap;
    time_t teraz_sec = time(NULL);
    struct tm teraz;
    va_start(ap, format);
    gmtime_r(&teraz_sec, &teraz);
    fprintf(stderr, "poczta [%d-%02d-%02d %02d:%02d:%02d]: ", 1900+teraz.tm_year, teraz.tm_mon,
                                                         teraz.tm_mday, teraz.tm_hour,
                                                         teraz.tm_min, teraz.tm_sec);
    vfprintf(stderr, format, ap);
    va_end(ap);
}

void zadaj_dostep(void) {
    struct sembuf s[3] = {{SEM_LL, 0, 0}, {SEM_LL, +1, 0}, {SEM_LK, 0, 0}};
    /* poczta potrzebuje dostep do zapisu, wiec ustawia semafor jakby byla listonoszem */
    semop(sem_id, s, 3);
}

void zwolnij_dostep(void) {
    struct sembuf s = {SEM_LL, -1, 0};
    semop(sem_id, &s, 1);
}

int rejestruj_skrzynke(void) { /* rejestruje skrzynke z kolejki wolnych i zwraca jej numer, lub -1 jesli brak skrzynek */
    long      num;
    struct msqid64_ds buf;
    msgctl(skrzynki_wolne, IPC_STAT, &buf);
```

```

    if (buf.msg_qnum==0)
        return -1;

    if (msgrcv(skrzynki_wolne, &num, 0, 0, 0) == -1) perror("rejestruj_skrzynke");
    num--; // patrz: num++ w zwolnij_skrzynke
    zadaj_dostepu();
    skrzynki[num].uzywana = 1;
    zwolnij_dostep();
    loguj("Zarejestrowano skrzynkę %d\n", num);
    return num;
}

int zwolnij_skrzynke(int num) { /* zwalnia skrzynke o numerze num i dodaje do kolejki wolnych */
    loguj("Zwalniam skrzynkę %d\n", num);
    zadaj_dostepu();
    skrzynki[num].uzywana = 0;
    skrzynki[num].lwiadomosci = 0;
    zwolnij_dostep();
    num++; // w kolejce przechowujemy numery skrzynek+1, bo 0 jest niepoprawne!
    if (msgsnd(skrzynki_wolne, &num, 0, 0) == -1) perror("zwolnij_skrzynke()");
    num--;
    return 0;
}

void serwuj(int kolejka) {
    int oczekujace, loczekujacych=0; /* oczekujace - kolejka komunikatow, ktorzych nie udalo sie wyslac */
    int tmp;
    zapytanie_t wiad, odp;
    oczekujace = msgget(IPC_PRIVATE, 0660);
    odp.nadawca = NADAWCA_POCZTA;
    odp.adr_zwrotny = kolejka;

    errno=0;
    while ((msgrcv(kolejka, &wiad, ROZM_TRESCI, 0, 0) != -1)&&(errno!=EINTR)) {
        /* tak dlugo jak nie trafisz na blad, lub nie otrzymasz sygnalu konca (SIGINT) lub innego przerwania */
        odp.operacja = wiad.operacja;
        loguj("operacja (OD:%d KOD:%d, ADR_ZWR:%d)\n", wiad.nadawca, wiad.operacja, wiad.adr_zwrotny);

        switch (wiad.operacja) {
            case (OPER_NOWA_SKRZYNKA):
                odp.numer = rejestruj_skrzynke();
                break;
            case (OPER_USUN_SKRZYNKE):
                odp.numer = zwolnij_skrzynke(wiad.numer);
                break;
            case (OPER_ILE_MIEJSCA):
                odp.numer = pojemnosc;
                break;
            case (OPER_ILE_SKRZYNEK):
                odp.numer = lskrzynek;
                break;
            case (OPER_ID_SEGMENTU):
                odp.numer = skr_id;
                break;
        }
        if (msgsnd(wiad.adr_zwrotny, &odp, ROZM_TRESCI, IPC_NOWAIT) == -1) { /* jesli nie dalo sie wyslac */
            switch (errno) {

```

```

    case (EAGAIN): /* brak miejsca w kolejce adresata, dodaj do kolejki oczekujacych */
        odp.adr_zwrotny = wiad.adr_zwrotny;
        msgsnd(oczekujace, &odp, ROZM_TRESCI, 0);
        loczekujacych++;
        break;
    case (EIDRM): /* adresat usunal kolejke, przed odebraniem potwierdzenia nadania - usun mu skrzynke */
        if (wiad.operacja == OPER_NOWA_SKRZYNKA) {
            zwolnij_skrzynke(odp.numer);
        }
        break;
    }
    loguj("odlozone (D0:%d, KOD:%d)\n", wiad.adr_zwrotny, odp.operacja);
}
if (loczekujacych>0) { /* jesli jest cos w kolejce oczekujacych */
    msgrcv(oczekujace, &odp, ROZM_TRESCI, 0, 0); /* odbierz to */
    tmp = odp.adr_zwrotny;
    odp.adr_zwrotny = kolejka;
    loguj("wysylam odlozone (D0:%d, KOD:%d)\n", tmp, odp.operacja);
    msgsnd(tmp, &odp, ROZM_TRESCI, 0); /* i wyslij ponownie, blokujac sie jesli tym razem znow zajete */
    loczekujacych--;
}
}

if (errno!=EINTR) { /* wystapil blad inny niz signal -> nie przewidziany */
    loguj("Blad odczytu kolejki\n");
    perror("msgrcv()");
}
msgctl(oczekujace, IPC_RMID, NULL);
}

int zajmij_zasoby(void) {
    int i;
    if ((skr_id = shmget(pocztaid, ROZM_SKRZYNEK, IPC_CREAT|0660)) == -1) {
        perror("shmget()");
        return FALSE;
    }
    if ((skrzynki = (skrzynka_t*) shmat(skr_id, NULL, 0)) == (void*)-1) {
        perror("shmat()");
        return FALSE;
    }
    if ((sem_id = semget(pocztaid, 3, IPC_CREAT|IPC_EXCL|0660)) == -1) {
        perror("semget()");
        return FALSE;
    }
    if (semctl(sem_id, SEM_LS, SETVAL, lskrzynek) == -1)
        perror("nie");
    if ((kolejka = msgget(pocztaid, IPC_CREAT|IPC_EXCL|0660)) == -1) {
        perror("msgget()");
        return FALSE;
    }
    if ((skrzynki_wolne = msgget(IPC_PRIVATE, 0660)) == -1) {
        perror("msgget()");
        return FALSE;
    }
}

for (i=0; i< lskrzynek; i++) {
    /* ustaw wskazniki do wiadomosci <-> sformatuj przestrzen skrzynek.

```

```

        UWAGA adres wiadomosci wzgledny, liczony od poczatku pamieci wspoldzielonej */
    skrzynki[i].wiadomosci = (wiadomosc_t*) ((lskrzynek*sizeof(skrzynka_t)) + i*(pojemnosc*sizeof(wiadomosc_t)));
    printf("%d offset: %d\n", i, skrzynki[i].wiadomosci);
    zwolnij_skrzynke(i); /* dodaje skrzynke do kolejki wolnych */
}
return TRUE;
}

void zwolnij_zasoby(void) {
    if (shmdt(skrzynki) == -1) {perror("shmdt()"); };
    if (shmctl(skr_id, IPC_RMID, NULL) == -1) {perror("shmctl(skr_id rm)"); };
    if (semctl(sem_id, 0, IPC_RMID, 0) == -1) {perror("semctl(sem rm)"); };
    if (msgctl(kolejka, IPC_RMID, NULL) == -1) {perror("msgctl(kol_id, rm)"); };
    if (msgctl(skrzynki_wolne, IPC_RMID, NULL) == -1) {perror("msgctl(skrzynki_wolne, rm)"); };
    loguj("Zasoby zwolnione\n");
}

void bezpieczne_wyjście(int sygnal) { /* wrapper sygnału */
    loguj("Otrzymano sygnał %d\n", sygnal);
    zwolnij_zasoby();
}

int main(int argc, char *argv[]) {
    if (argc<3) {
        printf("Używanie: poczta lskrzynek pojemnosc\n");
        return APP_FAILED;
    }
    if ((lskrzynek=atoi(argv[1]))==0) {
        printf("Liczba skrzynki musi być liczbą >0\n");
        return APP_FAILED;
    }
    if ((pojemnosc=atoi(argv[2]))==0) {
        printf("Pojemność musi być liczbą >0\n");
        return APP_FAILED;
    }

    pocztaid = ftok(SCIEZKAPOCZTA, (int) 'a');
    if (!zajmij_zasoby()) {
        return APP_FAILED;
    }
    signal(SIGINT, bezpieczne_wyjście);
    loguj("Poczta otwarta (SIGINT aby zamknąć)\n");

    serwuj(kolejka);

    if (errno!=EINTR) {
        loguj("Zwyczajne wyjście\n");
        zwolnij_zasoby();
    }
    return APP_SUCCESS;
}

```

### 3.3 listonosz.c

```

#include <stdio.h>
#include <stdlib.h> /* RAND_MAX */
#include <time.h> /* struct timespec */

```

```

#include <linux/ipc.h>
#include <linux/sem.h>
#include <linux/msg.h>
#include <linux/shm.h>
#include "struktury.h"

int pocztaid;

typedef struct wiad_kol {
    long nr_skrzynki;
    wiadomosc_t wiadomosc;
} wiad_kol_t;

void czekaj_chwile(int maks_sec) {
    struct timespec czas = {0, 0};
    czas.tv_sec = (int) (maks_sec*1.0*rand()/(RAND_MAX*1.0));
    nanosleep(&czas, NULL);
}

void zadaj_dostep(int sem_id) {
    struct sembuf s[3] = {{SEM_LL, 0, 0}, {SEM_LL, +1, 0}, {SEM_LK, 0, 0}};
    /* czekaj az liczba listonoszy=0, zwieksz liczbe listonoszy, czekaj az liczba klientow=0 */
    semop(sem_id, s, 3);
}

void zwolnij_dostep(int sem_id) {
    struct sembuf s = {SEM_LL, -1, 0}; /* zmniejsz liczbe listonoszy */
    semop(sem_id, &s, 1);
}

int w_partii(int partia) { /* ile jest wiadomosci w kolejce partia */
    struct msqid64_ds tmp;
    if (msgctl(partia, IPC_STAT, &tmp) == -1)
        perror("w_partii");
    else return tmp.msg_qnum;
}

int z_pliku_do_partii(FILE *plik, int partia) {
    /* skopiuj wiadomosci z pliku do partii, zakoncz, jesli napotkano koniec partii (nowa linia),
    lub napotkano koniec pliku. jesli nie napotkano konca pliku zwraca 1 (w przyszlosci bedzie
    jeszcze trzeba czytac z pliku bo cos tam jest), jesli napotkano koniec pliku zwraca 0 */
    char buf[144];
    wiad_kol_t wiad;
    int i;
    while ((!feof(plik)) && (fgets(buf, 144, plik)!=NULL) && (strlen(buf)>1)) {
        printf("Podniosłem wiadomość\n");
        memset(&wiad.wiadomosc, 0, sizeof(wiadomosc_t));
        sscanf(buf, "%d %[ -~]", &wiad.nr_skrzynki, &wiad.wiadomosc);
        wiad.nr_skrzynki++; /* wiadomosci w kolejce maja adres +1, zeby moc zaadresowac skrzynke nr. 0 */
        if (msgsnd(partia, &wiad, sizeof(wiadomosc_t), 0) == -1) { /* moze sie zdarzyc zapchanie partii */
            perror("z_pliku_do_pliku msgsnd");
        }
    }
    return !feof(plik);
}

int z_partii_do_partii(int zpartia, int dopartia) { /* przenies wiadomosci z zpartia do dopartia */

```



```

wiad_kol_t wiad;
while (msgrcv(zpartia, &wiad, sizeof(wiadomosc_t), 0, IPC_NOWAIT) != -1) {
    msgsnd(dopartia, &wiad, sizeof(wiadomosc_t), 0);
}
return TRUE;
}

int roznies_partie(skrzynka_t* skrzynki, int lskrzynek, int pojemnosc, int partia, int odlozone, int sem_id) {
    /* roznies wiadomosci do skrzynek z kolejki partia. jesli skrzynki zajete, przesyłaj wiadomosci
       do kolejki odlozone. gdy potrzebujesz dostep do skrzynek, uzywaj semafora sem_id */
    wiad_kol_t wiad;
    wiadomosc_t* adr;
    while (msgrcv(partia, &wiad, sizeof(wiadomosc_t), 0, IPC_NOWAIT) != -1) { /* poki sa wiadomosci */
        wiad.nr_skrzynki--; /* patrz komentarz: z_pliku_do_partii() o numeracji skrzynek w kolejkach */
        zadaj_dostep(sem_id);
        if (wiad.nr_skrzynki < lskrzynek && (skrzynki[wiad.nr_skrzynki].uzywana)) { /* jest tyle skrzynek i akurat ta jest u
            if (skrzynki[wiad.nr_skrzynki].lwiadomosci < pojemnosc) { /* jesli jest jeszcze miejsce, wstawiamy do skrzynki */
                printf("Wiadomość do skrzynki %d\n", wiad.nr_skrzynki);
                adr = (wiadomosc_t*) ((int)skrzynki + (int)skrzynki[wiad.nr_skrzynki].wiadomosci);
                memcpy(adr[skrzynki[wiad.nr_skrzynki].lwiadomosci++], wiad.wiadomosc, 128);
            } else { /* jesli nie ma miejsca, przenosimy do odlozonych */
                printf("Skrzynka %d: brak miejsca, prześlę później\n", wiad.nr_skrzynki);
                wiad.nr_skrzynki++;
                msgsnd(odlozone, &wiad, sizeof(wiadomosc_t), 0);
            }
            zwolnij_dostep(sem_id);
            czekaj_chwile(5);
        }
        else {
            zwolnij_dostep(sem_id);
            printf("Ehh, znowu zły nadawca, skrzynka %d nie jest używana\n", wiad.nr_skrzynki);
        }
    }
}

int main(int argc, char *argv[]) {
    int ta_partia, nast_partia; /* partia aktualnie roznoszona i partia odlozonych */
    int skr_id, sem_id; /* id pamieci wspoldzielonej, id semaforow */
    skrzynka_t* skrzynki; /* wskaznik do pamieci ze skrzynkami */
    int lskrzynek, pojemnosc; /* liczba skrzynek i ich pojemnosc */
    FILE* listy; /* plik z listami */

    if (argc < 2) {
        printf("Używanie: listonosz plik\n");
        return APP_FAILED;
    }

    srand(time(NULL)*getpid());

    pocztaid = ftok(SCIEZKAPOCZTA, (int) 'a');
    if (! poczta_otworz(pocztaid)) {
        printf("Nie można się dostać do poczty\n");
        return APP_FAILED;
    }

    if ((listy = fopen(argv[1], "r")) == NULL) {
        perror("fopen");
    }
}

```

```

    return APP_FAILED;
}

skr_id = zapytaj(NADAWCA_LISTONOSZ, OPER_ID_SEGMENTU, 0);
pojemnosc = zapytaj(NADAWCA_LISTONOSZ, OPER_ILE_MIEJSCA, 0);
lskrzynek = zapytaj(NADAWCA_LISTONOSZ, OPER_ILE_SKRZYNEK, 0);

sem_id = semget(pocztaid, 0, 0660);
if ((int)(skrzynki = (skrzynka_t*) shmat(skr_id, NULL, 0660)) == -1) {
    perror("shmat()");
    return APP_FAILED;
}

if ((ta_partia = msgget(IPC_PRIVATE, 0660)) == -1) {
    perror("msgget ta_partia");
    return APP_FAILED;
}

if ((nast_partia = msgget(IPC_PRIVATE, 0660)) == -1) {
    perror("msgget nast_partia");
    return APP_FAILED;
}

while (z_pliku_do_partii(listy, ta_partia) || w_partii(ta_partia)) {
    /* dopoki sa wiadomosci w pliku lub w partii wykonuj swe zadanie */
    printf("Idę roznieść partię listów\n");
    roznies_partie(skrzynki, lskrzynek, pojemnosc, ta_partia, nast_partia, sem_id);
    z_partii_do_partii(nast_partia, ta_partia);
    printf("Odpoczywam po rozniesieniu partii\n");
    czekaj_chwile(5);
}

msgctl(ta_partia, IPC_RMID, 0);
msgctl(nast_partia, IPC_RMID, 0);
poczta_zamknij();
return APP_SUCCESS;
}

```

### 3.4 klient.c

```

#include <stdio.h>
#include <stdlib.h> /* RAND_MAX */
#include <time.h> /* struct timespec */
#include <linux/ipc.h>
#include <linux/sem.h>
#include <linux/msg.h>
#include <linux/shm.h>
#include "struktury.h"

int pocztaid;

void czekaj_chwile(int maks_sec) {
    struct timespec czas = {0, 0};
    czas.tv_sec = (int) (maks_sec*1.0*rand()/(RAND_MAX*1.0));
    nanosleep(&czas, NULL);
}

```

```

void zadaj_dostep(int sem_id) {
    struct sembuf s[2] = {{SEM_LL, 0, 0}, {SEM_LK, +1, 0}};
    /* czekaj, az liczba listonoszy=0, zwieksz liczbe klientow o 1 */
    semop(sem_id, s, 2);
}

void zwolnij_dostep(int sem_id) {
    struct sembuf s = {SEM_LK, -1, 0}; /* zmniejsz liczbe klientow o 1 */
    semop(sem_id, &s, 1);
}

int czytaj_listy(skrzynka_t* skrzynka, wiadomosc_t* wiadomosci, int sem_id) {
    zadaj_dostep(sem_id);
    if (skrzynka->lwiadomosci>0) {
        do {
            printf("%s\n", wiadomosci[--skrzynka->lwiadomosci]);
        } while (skrzynka->lwiadomosci>0);
    } else
        printf("Brak wiadomości\n");
    zwolnij_dostep(sem_id);
}

int main(int argc, char *argv[]) {
    int skr_id, sem_id; /* identyfikatory pamieci wspoldzielonej i semaforow */
    int lprob, nr; /* liczba prob (podawana jako parametr), numer skrzynki */
    skrzynka_t* skrzynka; /* wskaznik do wlasnej skrzynki w obszarze pam.wspoldz. */
    wiadomosc_t* wiadomosci; /* bezwzgl. wskaznik do wiadomosci w skrzynce */
    if (argc<2) {
        printf("Uzywanie: klient lprob\n");
        return APP_FAILED;
    }

    srand(time(NULL)*getpid());

    pocztaid = ftok(SCIEZKAPOCZTA, (int) 'a');
    if (!poczta_otworz(pocztaid)) {
        printf("Nie można się dostać do poczty\n");
        return APP_FAILED;
    }

    if ((lprob = atol(argv[1])) == 0) {
        printf("lprob musi być liczbą >0\n");
        return APP_FAILED;
    }

    if ((nr = zapytaj(NADAWCA_KLIENT, OPER_NOWA_SKRZYNKA, 0)) != -1) {
        printf("Mam skrzynkę %d\n", nr);
    } else {
        printf("Nie mam skrzynki\n");
        return APP_FAILED;
    }

    skr_id = zapytaj(NADAWCA_KLIENT, OPER_ID_SEGMENTU, 0);
    sem_id = semget(pocztaid, 0, 0660);
    if ((int)(skrzynka = (skrzynka_t*) shmat(skr_id, NULL, 0660)) == -1) {
        perror("shmat()");
        return APP_FAILED;
    }
}

```

```

}
/* wskazniki do skrzynek w pamieci wspoldzielonej liczone sa od poczatkowego adresu tej pamieci,
   czyli wzglednie. zaleznie od zmapowania pam.wspoldz. kazdy program zmienia sobie wskaznik do
   wiadomosci o odp. przesuniecie (adres skrzynek) */
wiadomosci = (wiadomosc_t*) ((int)skrzynka + (int)skrzynka[nr].wiadomosci);
skrzynka = &skrzynka[nr];
80

while (lprob-->0) {
    printf("Idę na pocztę. . .\n");
    czytaj_listy(skrzynka, wiadomosci, sem_id);
    printf("Idę do domu\n");
    czekaj_chwile(5);
}

zapytaj(NADAWCA_KLIENT, OPER_USUN_SKRZYNKE, nr);
poczta_zamknij();
return APP_SUCCESS;
90
}

```

### 3.5 poczta\_dostep.h

```

#ifndef POCZTA_DOSTEP
#define POCZTA_DOSTEP

#ifndef _LINUX_IPC_H
#include <linux/ipc.h>
#endif

#ifndef STRUKTURY
#include "struktury.h"
10
#endif

int poczta;
int odp;

int poczta_otworz(int pocztaid);
int poczta_zamknij(void);
int zapytaj(int nadawca, int oper, int arg);

#endif
20

```

### 3.6 poczta\_dostep.c

```

#include "poczta_dostep.h"

int poczta_otworz(int pocztaid) { /* otworz kolejke do poczty i utworz wlasna kolejke */
    if (((poczta = msgget(pocztaid, 0660)) == -1) || ((odp = msgget(IPC_PRIVATE, 0660)) == -1)) {
        perror("msgget(poczta | odp)");
        return FALSE;
    } else
        return TRUE;
}

int poczta_zamknij(void) { /* usun wlasna kolejke */
    if (msgctl(odp, IPC_RMID, 0) == -1) {
        perror("msgctl(odp, rm)");
    }
10
}

```

```
    return FALSE;
} else
    return TRUE;
}
```

```
int zapytaj(int nadawca, int oper, int arg) { /* przeslij zapytanie i zwroc odpowiedz */
    zapytanie_t wiad = {nadawca, odp, oper, arg};
    if (msgsnd(poczta, &wiad, ROZM_TRESCI, 0) == -1) {
        perror("zapytaj snd");
        return FALSE;
    }
    if (msgrcv(odp, &wiad, ROZM_TRESCI, 0, 0) == -1) {
        perror("zapytaj rcv");
        return FALSE;
    }
    return wiad.numer;
}
```

20

30

## 4 Pliki

Kod źródłowy, program i wyniki działania znajdują się także pod adresem: <http://tri10o.republika.pl/so.shtml>