SOR - SYSTEMY OPERACYJNE

Środowisko

System: Debian GNU/Linux 13 (trixie), kernel 6.12.63
Język: C++17
Kompilator: g++ 14.2.0 (Debian 14.2.0-19)
Build system: CMake 3.31.6 (min. 3.10)
Biblioteki: pthreads (POSIX Threads)
Architektura: x86_64 (amd64)
Do uruchomienia wymagane nowsze lub kompatybilne

Generalny opis

Projekt realizuje symulacje Szpitalnego Oddziału Ratunkowego opierającego swoje działanie na nie scentralizowanym bez-busy-wait tworzeniu/zarządzaniu procesami oraz mechanizmami IPC. Projekt bazowo tworzy 10 procesów (1 dyrektor, 1 okienko rejestracji, 1 generator pacjentów, 7 lekarzy), korzysta z 1 pamięci dzielonej, z 13 semafor oraz 7 kolejek.

Flow projektu polega na wywołaniu dyrektora, który inicjuje wszystkie procesy (fork+exec) i mechanizmy IPC. Generator tworzy pacjentów, którzy:
- pojawiają się przed wejściem przed wejściem
- wchodzą do poczekalni (ograniczone miejscami)
- rejestrują się w okienku
- trafiają do lekarza POZ (ich stan jest weryfikowany)
- trafiają do konkretnego lekarza specjalisty (lekarz leczy/wystawia diagnozę, bierze pacjentów w gorszym stanie)
- symulacja obsługuje dodatkowo pacjentów dzieci (2 sloty w poczekalni), pacjentów VIP (szybsza rejestracja), dwa okienka rejestracji (kiedy w poczekalni za dużo ludzi), sygnały dyrektora (przerwa dla lekarza/ewakuacja całego SOR)

Kompilacja

1. Pobierz ZIP
2. `cd` do wypakowanego folderu (dalej jako /)
3. `mkdir build && cd build` - stwórz folder /build i wejdź do niego
4. `cmake ..` - zbuduj Makefile za pomocą CMake
5. `make -j$(nproc)` - skompiluj program
6. `cp dyrektor rejestracja lekarz pacjent generator ..` - skopiuj binarki z /build do /
7. `cd ..` - wyjdź do /
8. Program skompilowany :)

Uruchomianie

`./dyrektor` - uruchamia program z standardowymi parametrami (naturalna symulacja)
`./dyrektor -t 30` - uruchamia program, który zatrzyma się po 30sek (>0)
`./dyrektor -p 30` - program pozwoli na stworzenie maks 30 procesów (przynajmniej >11)
`./dyrektor -g 100 200` - program będzie generować pacjentów co 100ms-200ms (L<R)

Obsługa w trakcie działania

Klawisz: `1-6` - dyrektor wysyła odpowiedniego doktora na oddział (doktor nie bierze przez ten czas udziału w symulacji)
Klawisz: `7 / q` - ewakuacja SOR, graceful zamknięcie programu

Testy

Test 01 — Start i sprzątanie (test_01_startup_cleanup.sh)
Uruchamia ./dyrektor -t 5, czeka 2s, sprawdza czy procesy SOR żyją (≥10) i czy IPC (shm, sem, msg) istnieją. Po zakończeniu sprawdza czy wszystkie procesy zniknęły i IPC posprzątane

Test 02 - Limit procesów (test_02_process_limit.sh)
Uruchamia z -p 20 -t 10 -g 50 100 (szybkie generowanie, limit 20 procesów). Co sekundę liczy procesy przez 14s. Sprawdza czy limit nigdy nie został przekroczony i czy pacjenci w ogóle się pojawili

Test 03 - Stress test + zakleszczenia (test_03_stress_no_deadlock.sh)
Uruchamia z -g 10 20 (ekstremalnie szybkie generowanie) na 12s. Monitoruje 10s czy system działa. Po upływie czasu sprawdza czy dyrektor sam się zakończył (nie zakleszczyło się) i czy posprzątał procesy + IPC.

Test 04 - Zabicie potomnych (test_04_kill_children.sh)
Uruchamia z -t 12, czeka 3s, potem kill -9 na 2 lekarzy + rejestrację. Sprawdza czy dyrektor przeżył mimo utraty dzieci, czy sam się potem zakończył, i czy procesy + IPC zostały posprzątane.

Wszystkie testy (run_tests.sh)
Skrypt uruchamiający wszystkie 4 testy integracyjne po kolei. Przed startem buduje projekt (`cmake` + `make`). Między testami sprząta ewentualne pozostałe procesy SOR i zasoby IPC. Na końcu wypisuje podsumowanie: ile testów przeszło, ile nie, z kolorowym oznaczeniem (✓ / ✗). Użycie: `bash tests/run_tests.sh`

Wszystkie IPC

```
=== PAMIĘĆ DZIELONA (1 segment) ===
SharedState [ftok /tmp,'S'] — wspólny stan symulacji, dostęp chroniony SEM_SHM_MUTEX
  - start_time_sec/nsec    — timestamp startu (do logów)
  - shutdown               — flaga zakończenia (dyrektor ustawia, wszyscy czytają)
  - director_pid           — PID dyrektora
  - registration_pid       — PID rejestracji
  - doctor_pids[7]         — PID-y lekarzy (do wysyłania SIGUSR1/SIGUSR2)
  - doctor_on_break[7]     — flagi: czy lekarz jest na oddziale po SIGUSR1
  - reg_window_2_open      — czy okienko 2 jest otwarte
  - reg_queue_count        — ile osób w kolejce do rejestracji (decyduje o okienku 2)
  - total_patients         — ile pacjentów wygenerowano łącznie
  - patients_in_sor        — ile osób aktualnie w budynku (dziecko+opiekun=2)
  - active_patient_count — ile procesów pacjentów żyje (do limitu -p)
  - max_patients           — limit procesów (parametr -p, 0=brak)
  - specialist_msgids[7]   — ID kolejek specjalistów ([0]=-1, [1..6]=kolejki)
  - log_file[256]          — ścieżka do pliku logu

=== SEMAFORY (13 w jednym zestawie) ===
[ftok /tmp,'E']
 [0]  SEM_POCZEKALNIA              — wolne miejsca w poczekalni; -1 dorosły, -2 dziecko+opiekun
 [1]  SEM_REG_QUEUE_MUTEX          — mutex, chroni reg_queue_count w SharedState
 [2]  SEM_REG_WINDOW_1             — binary(1), stan okienka 1 (zawsze otwarte)
 [3]  SEM_REG_WINDOW_2             — binary(0), stan okienka 2 (dynamiczne otwarcie/zamknięcie)
 [4]  SEM_SPECIALIST_KARDIOLOG     — binary(1), 1 pacjent na raz u kardiologa
 [5]  SEM_SPECIALIST_NEUROLOG      — binary(1), 1 pacjent na raz u neurologa
 [6]  SEM_SPECIALIST_OKULISTA      — binary(1), 1 pacjent na raz u okulisty
 [7]  SEM_SPECIALIST_LARYNGOLOG — binary(1), 1 pacjent na raz u laryngologa
 [8]  SEM_SPECIALIST_CHIRURG       — binary(1), 1 pacjent na raz u chirurga
 [9]  SEM_SPECIALIST_PEDIATRA      — binary(1), 1 pacjent na raz u pediatry
[10]  SEM_SHM_MUTEX                — mutex, chroni cały SharedState przed race condition
[11]  SEM_LOG_MUTEX                — mutex, chroni zapis do pliku logu (1 proces pisze naraz)
[12]  SEM_REG_QUEUE_CHANGED        — event(0), budzi kontroler okienka 2 gdy kolejka się zmieni

=== KOLEJKI KOMUNIKATÓW (7 kolejek) ===
GŁÓWNA [ftok /tmp,'M'] — centralna komunikacja pacjent↔rejestracja↔POZ↔specjalista
  mtype=1                  — pacjent VIP → rejestracja (priorytet przez msgrcv -2)
  mtype=2                  — pacjent zwykły → rejestracja
  mtype=3                  — pacjent → lekarz POZ (triaż, FIFO)
  mtype=10000+id           — rejestracja → pacjent X (potwierdzenie rejestracji)
  mtype=20000+id           — POZ → pacjent X (kolor triażu + przypisany specjalista)
  mtype=30000+id           — specjalista → pacjent X (wynik: dom/oddział/inna placówka)

KARDIOLOG  [ftok /tmp,'a'] — pacjenci czekający na kardiologa, mtype=kolor (1=RED,2=YELLOW,3=GREEN)
NEUROLOG   [ftok /tmp,'b'] — j.w. dla neurologa
OKULISTA   [ftok /tmp,'c'] — j.w. dla okulisty
LARYNGOLOG [ftok /tmp,'d'] — j.w. dla laryngologa
CHIRURG    [ftok /tmp,'e'] — j.w. dla chirurga
PEDIATRA   [ftok /tmp,'f'] — j.w. dla pediatry (dzieci <18 trafiają TYLKO tutaj)
  → Lekarz odbiera msgrcv(-3) = najniższy mtype pierwszy = RED przed YELLOW przed GREEN
```

Techniczny opis projektu

=== CO SIĘ DZIEJE PO URUCHOMIENIU ./dyrektor ===

--- FAZA 1: INICJALIZACJA (main.cpp) ---
1. Dyrektor parsuje argumenty:
  -t sekundy — czas trwania symulacji (>0)
  -p maks_procesów — limit procesów łącznie (> FIXED_PROCESS_COUNT = 10)
  -g min_ms max_ms — interwał generowania pacjentów (oba >0, max >= min)
  Przy błędnych danych → komunikat + exit.

2. Ustaw handlery sygnałów (sigaction):
  SIGINT, SIGTERM, SIGHUP → ustawiają g_shutdown=1 i state->shutdown=1.
  Zarejestruj atexit(cleanupIPC) — sprzątanie IPC przy wyjściu.

3. initIPC() — tworzy wszystkie zasoby IPC:
  3a. PAMIĘĆ DZIELONA:
    ftok(/tmp,'S') → shmget(IPC_CREAT|IPC_EXCL|0600) → shmat() → memset(0).
    Rozmiar = sizeof(SharedState). Stary segment, jeśli istnieje, jest najpierw usuwany.
  3b. SEMAFORY (11 sztuk):
    ftok(/tmp,'E') → semget(SEM_COUNT=11, IPC_CREAT|IPC_EXCL|0600) →
semctl(SETALL):
    [0] SEM_POCZEKALNIA=20 — wolne miejsca
    [1] SEM_REG_QUEUE_MUTEX=1 — mutex kolejki
    [2] SEM_SPECIALIST_KARDIOLOG=1
    [3] SEM_SPECIALIST_NEUROLOG=1
    [4] SEM_SPECIALIST_OKULISTA=1
    [5] SEM_SPECIALIST_LARYNGOLOG=1
    [6] SEM_SPECIALIST_CHIRURG=1
    [7] SEM_SPECIALIST_PEDIATRA=1
    [8] SEM_SHM_MUTEX=1 — mutex pamięci
    [9] SEM_LOG_MUTEX=1 — mutex logów
    [10] SEM_REG_QUEUE_CHANGED=0 — event okienka 2
  3c. GŁÓWNA KOLEJKA:
    ftok(/tmp,'M') → msgget(IPC_CREAT|IPC_EXCL|0600)
  3d. 6 KOLEJEK SPECJALISTÓW:
    ftok(/tmp,'a'..'f') → msgget ×6, ID → state->specialist_msgids[1..6],
    slot [0] (POZ) = -1.

4. Wypełnij SharedState:
  start_time, director_pid, shutdown=0, max_patients,
  log_file="sor_log.txt", fopen("w").

--- FAZA 2: URUCHOMIENIE PROCESÓW ---
5. startRegistration():
  fork → prctl(PR_SET_PDEATHSIG,SIGTERM) → execl("./rejestracja"),
  PID → state->registration_pid, g_child_pids[].

6. startDoctors(): i=0..6:
   fork → prctl → execl("./lekarz", typ),
   typ: 0=POZ,1=kardio,2=neuro,3=okul,4=laryngo,5=chir,6=pedia,
   PID-y → state->doctor_pids[i], g_child_pids[].

7. setRawTerminal():
   tcgetattr, wyłącz ICANON i ECHO.

8. fork + execl("./generator"):
   -g → przekaż min_ms max_ms,
   PID → g_generator_pid, g_child_pids[].

--- FAZA 3: DZIAŁANIE PROCESÓW ---

=== REJESTRACJA (rejestracja.cpp) ===
9. Podłącza IPC: shmget, shmat, semget, msgget.

10. Tworzy wątki:
    queueControllerThread, windowThread (okienko 2).

11. Okienko 1 (wątek główny):
    msgrcv(mtype=-2) → VIP(1) przed zwykłym(2),
    processPatient(): reg_queue_count--,
    semSignal(SEM_REG_QUEUE_CHANGED),
    randomSleep(250–500ms),
    msgsnd(10000+id).

12. Kontroler kolejki:
    semWait(SEM_REG_QUEUE_CHANGED),
    queue>=10 i zamknięte → otwórz + pthread_cond_signal,
    queue<7 i otwarte → zamknij,
    pthread_kill(SIGUSR1) aż wyjdzie z msgrcv.

13. Okienko 2:
    pthread_cond_wait,
    po starcie jak okienko 1,
    g_window2_should_run=false → czekaj.

=== LEKARZE (lekarz.cpp) ===
14. Lekarz:
    IPC + sigaction:
    SIGUSR1→g_go_to_ward,
    SIGUSR2/SIGTERM/SIGINT→g_shutdown,

15. POZ (typ=0):
    msgrcv(mtype=3),
    randomSleep(1000–2000ms),
    triage: 10% RED, 35% YELLOW, 50% GREEN, 5% OUT,

OUT → msgsnd(20000+id,POZ),
IN → losuj specjalistę (dzieci→pedia),
POZ ignoruje SIGUSR1.

16. Specjaliści (1..6):
msgrcv(mtype=-3),
semWait,
randomSleep(2000–3000ms),
outcome: 85% dom, 14.5% oddział, 0.5% inna,
msgsnd(30000+id),
semSignal,
SIGUSR1 → goToWard():
doctor_on_break[X]=1,
sleep(250–500ms),
doctor_on_break[X]=0.

=== GENERATOR (generator.cpp) ===
17. IPC, args [min_ms max_ms] (def 500–1000),
SIGTERM/SIGINT→g_gen_shutdown.

18. Pętla:
randomSleep,
limit -p → czekaj active_patient_count <
(max_patients-FIXED_PROCESS_COUNT),
losuj wiek,VIP,
++total_patients, ++active_patient_count,
fork→execl("./pacjent"),
waitpid(WNOHANG).

=== PACJENT (pacjent.cpp) ===
19. Args, IPC,
SIGUSR2/SIGTERM/SIGINT→g_shutdown.

20. DOROSŁY (≥18):
enterWaitingRoom(): semWait, patients_in_sor++,
doRegistration(): reg_queue_count++, semSignal, msgsnd, msgrcv(10000+id),
doTriage(): msgsnd(3), msgrcv(20000+id),
doSpecialist(): msgsnd(kolor), msgrcv(30000+id),
exitSOR(): patients_in_sor--, active_patient_count--, semSignal.

21. DZIECKO (<18):
Rodzic: semop(-2), doRegistration(), pthread_cond_signal,
Dziecko: pthread_cond_wait, doTriage(), doSpecialist(),
join + semop(+2).

--- FAZA 4: PĘTLA DYREKTORA ---

22. handleKeyboard():
    '1'–'6'→SIGUSR1,
    '7'→SIGUSR2 (ewakuacja),
    'q'→shutdown,

--- FAZA 5: ZAMYKANIE ---
23. restoreTerminal().

24. state->shutdown=1.

25. Generator:
    SIGTERM, waitpid do 5s,
    potem SIGKILL.

26. Lekarze + rejestracja:
    SIGTERM → SIGKILL → waitpid.

27. cleanupIPC():
    shmctl, semctl, msgctl.

28. ipcs czyste.

--- FAZA AWARYJNA ---
29. PR_SET_PDEATHSIG → SIGTERM do potomków.

30. Rejestracja:
    kill(director_pid,0) → sprząta IPC.

LINKI DO KODU

Tworzenie i obsługa plików/odczyt:
- sor_common.hpp:427 - dopisywanie do logu fopen()/fprint()/fclose()
- main.cpp:544 - odczyt klawiatury read()

Tworzenie procesów:
- generator.cpp:118 - tworzenie pacjentów fork()/exec()
- main.cpp:460 - tworzenie lekarzy fork()/exec()
- main.cpp:200 - czyszczenie po dyrektorze waitpid()

Tworzenie i obsługa wątków:
- pacjent.cpp:110 - tworzenie/sprzątanie wątków dziecko + rodzic
- rejestracja.cpp:97 - obsługa wątku okienka rejestracji 2

Obsługa sygnałów:
- main.cpp:636 - obsługa sygnałów w main
- main.cpp:566 - ewakuacja lekarzy SIGUSR2

Synchronizacja procesów:
- sor_common:541 - tworzenie klucza IPC ftok()
- main.cpp:302 - tworzenie semafor semget()
- main.cpp:330 - ustawianie wartości semctl()
- sor_common:355 - blokujące semop()

Segment pamięci dzielonej:
- main.cpp:278 - tworzenie segmentu SharedState
- main.cpp:284 - podłączenie shmat()
- main.cpp:384 - odłączenie shmdt()
- main.cpp:390 - czyszczenie semctl()

Kolejki komunikatów:
- main.cpp:344 - główna kolejka msgget()
- pacjent.cpp:306 - prośba o rejestracje msgsnd()
- lekarz.cpp:178 - lekarz POZ odbiera msgrcv()
- main.cpp:402 - czyszczenie głównej kolejki msgctl()

Obsługa wejścia -t -p -g
- main.cpp:65 - obsługa całości

Testy
- /tests

Customowa obsługa błędów (strerror, errno)
- sor_common.hpp:289 - centralna funkcja
- sor_common.hpp:334 - definy