Raport - T7 Michał Kalinowski

alb. 151883

Projekt Systemy Operacyjne

Politechnika Krakowska, Informatyka NST, III semestr Michał Kalinowski

1. Cel i założenia projektowe

Celem projektu jest odwzorowanie zachowań owadów w ulu przy użyciu mechanizmów systemu operacyjnego:

- Współdzielenie zasobów (pamięć współdzielona, semafory).
- Wielowątkowość: pszczoły-robotnice w osobnych wątkach.
- Ograniczona pojemność ula (capacity) – jednocześnie tylko określona liczba pszczół/jaj.
- Możliwość dynamicznej zmiany pojemności ula przez pszczelarza (poprzez sygnały SIGUSR1 / SIGUSR2).
- Symulacja cyklu życiowego pszczół (pszczoła „umiera" po ustalonej liczbie wizyt w ulu).
- Wieloprocesowość: tworzenie procesów (królowa queen, wylęgający hatch).

Kluczowe założenia:

- Dwa procesy potomne:
  - queen – składa jaja (jeśli w ulu jest miejsce).
  - hatch – wylęga pszczoły z jaj.
- Wątek „pszczelarza" (listening na sygnały SIGUSR1 i SIGUSR2).
- N (zadana liczba) wątków „robotnic" na starcie.
- Pamięć współdzielona (System V) trzyma strukturę EggQueue z informacjami:
  - -occupant_count (aktualna liczba pszczół + jaj)
  - -capacity (maks. liczba rezydentów),
  - -kolejka jaj do wylęgania.

- Semafory:
  - Posix: ul_wejscie (liczbowe ograniczenie dostępności miejsc w ulu),
  - wejscie1_kierunek, wejscie2_kierunek (kontrola wchodzenia/wychodzenia),
  - SysV: do blokowania dostępu do kolejki jaj (sem_id).
- Obsługa sygnałów SIGUSR1, SIGUSR2, SIGINT (zatrzymanie programu).

- Kontrola:
  - Pszczoły wchodzą do ula, śpią tam określoną liczbę sekund, wychodzą, śpią poza ulem i tak w kółko, aż visits_left wyczerpie się.
  - Królowa co pewien czas próbuje złożyć jajo (jeśli nie ma miejsca, czeka).
  - Proces wylęgania co pewien czas „wyjmuje" jajo i tworzy nowy wątek pszczoły.
  - Wątek pszczelarza pozwala powiększyć/zmniejszyć ul za pomocą sygnałów.

2. Opis ogólny kodu (architektura)

Kod podzielony jest na kilka plików:

- main.c – punkt startowy, czyta N i P, inicjuje pamięć współdzieloną, semafory, uruchamia procesy queen i hatch, wątek pszczelarza oraz startowe wątki pszczół. Obsługuje również SIGINT.
- bee.[ch] – logika tworzenia i życia pszczół (wątki).
- queen.[ch] – proces queen (składanie jaj) i hatch (wylęganie).
- hive.[ch] – logika „wejścia/wyjścia" z ula, wyświetlanie stanu ula, zmiana pojemności
- egg.[ch] – struktura i operacje na kolejce jaj (EggQueue), zarządzanie pamięcią współdzieloną, semafor System V.
- beekeeper.[ch] – wątek pszczelarza, odbieranie sygnałów SIGUSR1 / SIGUSR2 i wywołanie adjust_hive_capacity().
- cleanup.[ch] – funkcja sprzątająca (zamyka procesy, wątki i zwalnia zasoby).
- error_handling.[ch] – definicje błędów i funkcja handle_error().

Co się udało zrobić:

- Działa poprawne tworzenie procesów i wątków: queen, hatch, bee.

- Bezpieczne zarządzanie tablicą wątków pszczół (z powiększaniem realloc).
- Obsługa pamięci współdzielonej, semaforów POSIX i SysV.
- Dynamiczna modyfikacja capacity (sygnały SIGUSR1, SIGUSR2).
- Mechanizm ograniczający kilkukrotną nieudaną próbę utworzenia wątku (jeśli errno == EAGAIN).
- Kolorowe logi i wyświetlanie stanu ula.

Z czym były problemy:

- Kolejność logów (asynchronizacja wątków i procesów) powoduje „przeplatanie" komunikatów.
- Zmniejszenie capacity nie wyrzuca pszczół już znajdujących się w ulu – część pszczół może więc przekroczyć nowy limit do czasu, aż same wyjdą.
- „Ostatnie pszczoły" mogą jeszcze chwilę działać po wywołaniu cleanup(), bo w praktyce czekamy aż wszystkie wątki się zakończą (anulowanie i pthread_join).

Linki do fragmentów kodu:

- **Tworzenie procesów:**

fork():

https://github.com/Elleass/SO-Projekt/blob/1df356fa0ab6aa656ef2e6ecd1eddb7ce0083702/src/main.c#L158C4-L158C24

wait():

https://github.com/Elleass/SO-Projekt/blob/1df356fa0ab6aa656ef2e6ecd1eddb7ce0083702/src/cleanup.c#L45C9-L45C37

exit():

https://github.com/Elleass/SO-Projekt/blob/1df356fa0ab6aa656ef2e6ecd1eddb7ce0083702/src/bee.c#L63C14-L63C36

- Tworzenie i obsługa wątków:

pthread_create():

https://github.com/Elleass/SO-Projekt/blob/1df356fa0ab6aa656ef2e6ecd1eddb7ce0083702/src/bee.c#L47C5-L47C52

pthread_join():

https://github.com/Elleass/SO-Projekt/blob/1df356fa0ab6aa656ef2e6ecd1eddb7ce0083702/src/cleanup.c#L74C9-L74C44

pthread_mutex_lock():

https://github.com/Elleass/SO-Projekt/blob/1df356fa0ab6aa656ef2e6ecd1eddb7ce0083702/src/cleanup.c#L66

pthread_mutex_unlock():

https://github.com/Elleass/SO-Projekt/blob/1df356fa0ab6aa656ef2e6ecd1eddb7ce0083702/src/cleanup.c#L80C1-L80C43

- Obsługa sygnałów:

sigaction()

https://github.com/Elleass/SO-Projekt/blob/1df356fa0ab6aa656ef2e6ecd1eddb7ce0083702/src/main.c#L90C1-L90C44

https://github.com/Elleass/SO-Projekt/blob/1df356fa0ab6aa656ef2e6ecd1eddb7ce0083702/src/beekeeper.c#L44

kill()

https://github.com/Elleass/SO-Projekt/blob/1df356fa0ab6aa656ef2e6ecd1eddb7ce0083702/src/cleanup.c#L44C8-L44C34

- Semafory SysV:

semget()

https://github.com/Elleass/SO-Projekt/blob/1df356fa0ab6aa656ef2e6ecd1eddb7ce0083702/src/egg.c#L33C5-L33C70

semctl()

https://github.com/Elleass/SO-Projekt/blob/1df356fa0ab6aa656ef2e6ecd1eddb7ce0083702/src/egg.c#L61

semop()

https://github.com/Elleass/SO-Projekt/blob/1df356fa0ab6aa656ef2e6ecd1eddb7ce0083702/src/egg.c#L151

- Semafory POSIX:

seminit()

https://github.com/Elleass/SO-Projekt/blob/1df356fa0ab6aa656ef2e6ecd1eddb7ce0083702/src/main.c#L134C1-L143C39

sem_trywait()

https://github.com/Elleass/SO-Projekt/blob/1df356fa0ab6aa656ef2e6ecd1eddb7ce0083702/src/hive.c#L31C8-L34C14

- Obsługa pamięci współdzielonej:

shmget()

https://github.com/Elleass/SO-Projekt/blob/1df356fa0ab6aa656ef2e6ecd1eddb7ce0083702/src/egg.c#L33

shmctl()

https://github.com/Elleass/SO-Projekt/blob/1df356fa0ab6aa656ef2e6ecd1eddb7ce0083702/src/egg.c#L74C1-L74C40

- Obsługa błędów

perror()

https://github.com/Elleass/SO-Projekt/blob/1df356fa0ab6aa656ef2e6ecd1eddb7ce0083702/src/main.c#L92

errno() (EAGAIN)

https://github.com/Elleass/SO-Projekt/blob/1df356fa0ab6aa656ef2e6ecd1eddb7ce0083702/src/bee.c#L55

Testy:

1. Test podstawowy

Cel: Sprawdzenie, czy program uruchamia się poprawnie dla przykładowych danych wejściowych i wykonuje podstawowe zadania
Przykładowe parametry wejściowe:
-N=6 (łącznie pszczół “robotnic”)
-P=2(początkowa maksymalna pojemność ula, P<N/2P<N/2)

```
 Podaj całkowitą liczbę pszczół w roju (N) :  6
 Podaj maksymalną liczbę pszczół w ulu (P), gdzie P < N/2: 2
 Maksymalna liczba pszczół w ulu: 2
 inside queen_process (child PID: 125373)
 utworzono hatch_eggs (child PID: 125374)
 [Beekeeper] Oczekiwanie na sygnały:
   - SIGUSR1 (kill -USR1 125356) => powiększ ul do min(2*N, cap
acity*2)
   - SIGUSR2 (kill -USR2 125356) => zmniejsz ul do capacity/2
 Pszczoła 0 startuje w wątku.
 Pszczoła 0:  wchodzi do ula wejściem 1.
 Wolna przestrzeń: 1 (zajęte: 1). Bees: 1, Eggs: 0
 Pszczoła 1 startuje w wątku.
 Pszczoła 1:  wchodzi do ula wejściem 1.
 Ul jest pełny. | Bees: 2, Eggs: 0
 Pszczoła 2 startuje w wątku.
 Pszczoła 2:  czeka - ul pełny.
 szczoła 3:  czeka - ul pełny.
 Pszczoła 4:  czeka - ul pełny.
 Pszczoła 0:  wychodzi z ula wejściem 1.
 Wolna przestrzeń: 1 (zajęte: 1). Bees: 1, Eggs: 0
 Pszczoła 2:  wchodzi do ula wejściem 1.
 Ul jest pełny. | Bees: 2, Eggs: 0
 Pszczoła 3:  czeka - ul pełny.
 Pszczoła 5:  czeka - ul pełny.
 Pszczoła 1:  wychodzi z ula wejściem 1.
 Wolna przestrzeń: 1 (zajęte: 1). Bees: 1, Eggs: 0
 Pszczoła 4:  wchodzi do ula wejściem 1.
 Ul jest pełny. | Bees: 2, Eggs: 0
 Pszczoła 3:  czeka - ul pełny.
 Pszczoła 5:  czeka - ul pełny.
 Krolowa: Ul jest pelny, nie mozna zlozyc jaj.
 Pszczoła 3:  czeka - ul pełny.
```

Zgodnie z założeniem pszczoły czekają aż miejsce w ulu się zwolni, królowa w przypadku pełnego ula nie może złożyć jaj.

2. Test wysokich wartości (ponad maksymalną wartość wątków)

Cel: Sprawdzenie czy w przypadku przekroczenia limitu dopuszczalnych wątków program wyłączy się w sposób przewidywalny.

```
...
Failed to create bee thread (errno=11)
Failed to create bee thread (errno=11)
Failed to create bee thread (errno=11)
Repeated thread creation failures. Stopping program.
Pszczoła 4448:  wchodzi do ula wejściem 1.
Wolna przestrzeń: 3526 (zajęte: 474). Bees: 474, Eggs: 0
Pszczoła 4449:  czeka, oba wejścia są zajęte.
Pszczoła 4214 startuje w wątku.
Pszczoła 4214 kończy życie.
Pszczoła 4216 startuje w wątku.
Pszczoła 4216 kończy życie.
Pszczoła 4452 startuje w wątku.
Pszczoła 4452 kończy życie.
Pszczoła 4032 startuje w wątku.
Pszczoła 4032 kończy życie.
Pszczoła 4453:  czeka, oba wejścia są zajęte.
Pszczoła 4218 startuje w wątku.
...
Queen process terminated.
Terminating hatch process (PID: 131278)...
Hatch process terminated.
Canceling beekeeper thread...
Beekeeper thread terminated.
.
Pszczoła 4623 kończy życie.
Pszczoła 4041 startuje w wątku.
Pszczoła 4041 kończy życie.
Pszczoła 4655 kończy życie.
Bee threads cleaned up.
Destroying semaphores...
Semaphores destroyed.
Destroying egg queue...
Egg queue destroyed.
Cleanup completed. Exiting.
```

Po trzykrotnym wywołaniu się: Failed to create bee thread (errno=11) program wciąż tworzy zakolejkowane pszczoły które od razu umierają, dzieje się tak ponieważ program wpierw musi zamknąć procesy queen oraz hatch które mogły by tworzyć

nowe pszczoły, następnie wykonuje się ciąg dalszy funkcji cleanup co pozwala zamknąć program w pełni kontrolowany sposób.

Test obsługi sygnałów SIGUSR1(zwiększenie pojemności) , SIGUSR2(zmniejszenie pojemności)

Parametry początkowe: N=10,P=4.
Cel: Sprawdzić, czy po wysłaniu sygnału SIGUSR1 do procesu głównego zwiększa się capacity, co umożliwia wejście większej liczbie pszczół oraz analogicznie po wysłaniu SIGUSR2 do procesu głównego zmniejsz się capacity o 50%.

```
Ul jest pełny. | Bees: 4, Eggs: 0
Pszczoła 2:  czeka - ul pełny.
Pszczoła 3:  czeka - ul pełny.
Zwiększanie pojemności ula z 4 do 8
Pszczoła 0:  wchodzi do ula wejściem 1.
Wolna przestrzeń: 3 (zajęte: 5). Bees: 5, Eggs: 0
Pszczoła 3:  wchodzi do ula wejściem 1.
Wolna przestrzeń: 2 (zajęte: 6). Bees: 6, Eggs: 0
Pszczoła 2:  wchodzi do ula wejściem 1.
Wolna przestrzeń: 1 (zajęte: 7). Bees: 7, Eggs: 0
Pszczoła 8:  wychodzi z ula wejściem 1.
Wolna przestrzeń: 2 (zajęte: 6). Bees: 6, Eggs: 0
Pszczoła 9:  wychodzi z ula wejściem 1.
Wolna przestrzeń: 3 (zajęte: 5). Bees: 5, Eggs: 0
Pszczoła 6:  wychodzi z ula wejściem 1.
Wolna przestrzeń: 4 (zajęte: 4). Bees: 4, Eggs: 0
Pszczoła 0 kończy życie.
Pszczoła 7:  wchodzi do ula wejściem 1.
Wolna przestrzeń: 3 (zajęte: 5). Bees: 5, Eggs: 0
Pszczoła 5 kończy życie.
Pszczoła 1:  wchodzi do ula wejściem 1.
Wolna przestrzeń: 2 (zajęte: 6). Bees: 6, Eggs: 0
Krolowa: Zlozono jajo ID: 4 (wykluje sie za 5 s)
Pszczoła 2:  wychodzi z ula wejściem 1.
Wolna przestrzeń: 2 (zajęte: 6). Bees: 5, Eggs: 1
```

Po wysłaniu komendy kill -USR1 <PID> w innym terminalu ul powiększył się dwukrotnie i od razu wypełnił się czekającymi pszczołami.

```
Zmniejszanie pojemności ula z 8 do 4
Pszczoła 2:  wychodzi z ula wejściem 1.
```

```
Ul jest pełny. | Bees: 5, Eggs: 0
Krolowa: Zlozono jajo ID: 8 (wykluje sie za 5 s)
Krolowa: Ul jest pelny, nie mozna zlozyc jaj.
Pszczoła 4:  wychodzi z ula wejściem 1.
Ul jest pełny. | Bees: 4, Eggs: 1
Pszczoła 6:  wychodzi z ula wejściem 1.
Ul jest pełny. | Bees: 3, Eggs: 1
Wykluwanie: Jajko ID: 8 wyklulo sie w pszczole!
Pszczoła 8 startuje w wątku.
Pszczoła 8:  wchodzi do ula wejściem 1.
Ul jest pełny. | Bees: 4, Eggs: 0
Krolowa: Ul jest pelny, nie mozna zlozyc jaj.
Pszczoła 2:  wchodzi do ula wejściem 1.
Ul jest pełny. | Bees: 5, Eggs: 0
Pszczoła 5:  wychodzi z ula wejściem 1.
Ul jest pełny. | Bees: 4, Eggs: 0
Pszczoła 3:  wychodzi z ula wejściem 1.
Wolna przestrzeń: 1 (zajęte: 3). Bees: 3, Eggs: 0
Pszczoła 4:  wchodzi do ula wejściem 1.
Ul jest pełny. | Bees: 4, Eggs: 0
Pszczoła 7:  wychodzi z ula wejściem 1.
...
Pszczoła 10:  wchodzi do ula wejściem 1.
Wolna przestrzeń: 1 (zajęte: 3). Bees: 3, Eggs: 0
Pszczoła 8:  wychodzi z ula wejściem 1.
Wolna przestrzeń: 2 (zajęte: 2). Bees: 2, Eggs: 0
Pszczoła 9:  wchodzi do ula wejściem 1.
Wolna przestrzeń: 1 (zajęte: 3). Bees: 3, Eggs: 0
Pszczoła 7:  wchodzi do ula wejściem 1.
Ul jest pełny. | Bees: 4, Eggs: 0
```

Po wysłaniu komendy kill -USR2 <PID> w innym terminalu ul zmniejszył się dwukrotnie jednak od razu nie "wyrzucił" pszczół z ula oraz wpuścił ponad limit pszczoły czekające w kolejce przed wysłaniem komunikatu.
Napotkany problem:
Program nie wyrzuca pszczół które już znajdowały się w ulu przed zmniejszeniem pojemności, ani nie blokuje tych, które były w trakcie wejścia. W efekcie przez jakiś czas może być w ulu więcej pszczół niż nowa, zredukowana wartość miejsc.

Test zamykania programu Ctrl + C

```
Krolowa: Ul jest pelny, nie mozna zlozyc jaj.
Krolowa: Ul jest pelny, nie mozna zlozyc jaj.
^C
SIGINT received. Stopping...
Starting cleanup...
Terminating queen process (PID: 138998)...
Queen process terminated.
Terminating hatch process (PID: 138999)...
Hatch process terminated.
Canceling beekeeper thread...
Beekeeper thread terminated.
Canceling 10 bee threads...
Bee threads cleaned up.
Destroying semaphores...
Semaphores destroyed.
Destroying egg queue...
Egg queue destroyed.
Cleanup completed. Exiting.
```

Po wysłaniu sygnału SIGINT program zamyka wszystkie procesy, czyści utworzone wątki oraz kolejke.

Test krótkiej żywotności pszczół

pszczoła ginie po pierwszym wyjściu z ula. Cel: sprawdzenie , czy occupant_count jest poprawnie dekrementowany i czy wątki „umierają” właściwie.

```
Pszczoła 9:  wychodzi z ula wejściem 1.
Wolna przestrzeń: 1 (zajęte: 3). Bees: 3, Eggs: 0
Pszczoła 8:  wychodzi z ula wejściem 1.
Wolna przestrzeń: 2 (zajęte: 2). Bees: 2, Eggs: 0
Pszczoła 5:  wychodzi z ula wejściem 1.
Wolna przestrzeń: 3 (zajęte: 1). Bees: 1, Eggs: 0
Pszczoła 6:  wychodzi z ula wejściem 1.
Wolna przestrzeń: 4 (zajęte: 0). Bees: 0, Eggs: 0
Pszczoła 0:  wchodzi do ula wejściem 1.
Wolna przestrzeń: 3 (zajęte: 1). Bees: 1, Eggs: 0
Pszczoła 4:  wchodzi do ula wejściem 1.
Wolna przestrzeń: 1 (zajęte: 3). Bees: 3, Eggs: 0
Pszczoła 7:  wchodzi do ula wejściem 2.
Wolna przestrzeń: 1 (zajęte: 3). Bees: 3, Eggs: 0
```

```
Pszczoła 0:  kończy życie.
Pszczoła 3:  kończy życie.
Pszczoła 1:  kończy życie.
Pszczoła 2:  kończy życie.
Pszczoła 0:  wychodzi z ula wejściem 1.
Wolna przestrzeń: 2 (zajęte: 2). Bees: 2, Eggs: 0
Pszczoła 4:  wychodzi z ula wejściem 1.
Wolna przestrzeń: 3 (zajęte: 1). Bees: 1, Eggs: 0
Pszczoła 7:  wychodzi z ula wejściem 1.
Wolna przestrzeń: 4 (zajęte: 0). Bees: 0, Eggs: 0
Pszczoła 9:  kończy życie.
Pszczoła 8:  kończy życie.
Pszczoła 5:  kończy życie.
Pszczoła 6:  kończy życie.
Krolowa: Zlozono jajo ID: 1 (wykluje sie za 5 s)
Wykluwanie: Jajko ID: 1 wyklulo sie w pszczole!
Pszczoła 1:  startuje w wątku.
Pszczoła 1:  wchodzi do ula wejściem 1.
Wolna przestrzeń: 3 (zajęte: 1). Bees: 1, Eggs: 0
Pszczoła 0:  kończy życie.
Pszczoła 4:  kończy życie.
Pszczoła 7:  kończy życie.
```

occupant_count spada do zera, gdy wszyscy wyjdą.
Pszczoły z krótkim życiem nie powracają do ula drugi raz.
Królowa i proces wylęgania nadal mogą generować nowe pszczoły (z nowymi ID).
Napotkany problem: Kolejność pojawiania się logów nie jest precyzyjna przez co zachowanie programu może stawać się nie czytelne. Zanim krolowa zlozy pierwsze jajo pszczoly umieraja przez co krolowa tworzy pszczole z ID martwej pszczoly.

6. Podsumowanie

- Projekt pozwala na sprawdzenie mechanizmów synchronizacji i komunikacji międzyprocesowej (pamięć współdzielona, semafory, sygnały).
- Udało się zrealizować wszystkie główne założenia: proces królowej i wylęgania, wątki pszczół, wątek pszczelarza reagujący na sygnały.
- Obsługa wielu wątków i procesów generuje asynchronię w logach, co jest normalne w aplikacjach wielowątkowych.

- Główne ograniczenie to brak wyrzucania pszczół z ula, jeśli zmniejszymy capacity „w locie”. Można to uznać za rozszerzenie na przyszłość.
- Testy pokazały, że mechanizmy zatrzymania (SIGINT) i sprzątania zasobów działają stabilnie