Systemy operacyjne

Lista zadań nr 10

Na zajęcia 17 grudnia 2020

Należy przygotować się do zajęć czytając następujące rozdziały książek:

- Advanced Programming in the UNIX Environment (wydanie trzecie): 11.1 – 11.5, 12.8 – 12.10
- Arpaci-Dusseau: Concurrency: An Introduction[1], Interlude: Thread API[2], Event-based Concurrency[3]

UWAGA! W trakcie prezentacji należy być gotowym do zdefiniowania pojęć oznaczonych **wytłuszczoną** czcionką.

Zadanie 1. Czym różni się **przetwarzanie równoległe** (ang. *parallel*) od **przetwarzania współbieżnego** (ang. *concurrent*)? Czym charakteryzują się **procedury wielobieżne** (ang. *reentrant*)? Podaj przykład procedury w języku `C` (a) wielobieżnej, ale nie **wątkowo-bezpiecznej** (ang. *thread-safe*) (b) na odwrót. Kiedy w jednowątkowym procesie uniksowym może wystąpić współbieżność?

Zadanie 2. Wybierz odpowiedni scenariusz zachowania wątków, w którym konkurują o dostęp do zasobów, i na tej podstawie precyzyjnie opisz zjawisko **zakleszczenia** (ang. *deadlock*), **uwięzienia** (ang. *livelock*) oraz **głodzenia** (ang. *starvation*). Dalej rozważmy wyłącznie ruch uliczny. Kiedy na jednym lub wielu skrzyżowaniach może powstać każde z tych zjawisk? Zaproponuj metodę (a) **wykrywania i usuwania** zakleszczeń (b) **zapobiegania** zakleszczeniom.

Wskazówka: Określ zbiór właściwości $\{p_i\}$, które zachodzą: zawsze, nigdy, od pewnego momentu t_k, nieskończenie wiele razy.

Dla odważnych: Spróbuj użyć logiki LTL[4], o której więcej będzie na przedmiocie „Automated verification".

Zadanie 3. W poniższym programie występuje **sytuacja wyścigu** (ang. *race condition*) dotycząca dostępów do współdzielonej zmiennej «`tally`». Wyznacz jej najmniejszą i największą możliwą wartość.

```
1 const int n = 50;
2 shared int tally = 0;
3
4 void total() {
5   for (int count = 1; count <= n; count++)
6     tally = tally + 1;
7 }
8
9 void main() { parbegin (total(), total()); }
```

Dyrektywa «`parbegin`» rozpoczyna współbieżne wykonanie procesów. Maszyna wykonuje instrukcje arytmetyczne wyłącznie na rejestrach – tj. kompilator musi załadować wartość zmiennej «`tally`» do rejestru, przed wykonaniem dodawania. Jak zmieni się przedział możliwych wartości zmiennej «`tally`», gdy wystartujemy k procesów zamiast dwóch? Odpowiedź uzasadnij pokazując przeplot, który prowadzi do określonego wyniku.

Zadanie 4. Podaj odpowiedniki funkcji `fork(2)`, `exit(3)`, `waitpid(3)`, `waitpid(2)`, `atexit(3)` oraz `abort(3)` dla wątków i opisz ich semantykę. Niepogrzebane procesy określamy mianem zombie – aby zapobiec ich powstawianiu ustawiamy dyspozycję sygnału «`SIGCHLD`» na «`SIG_IGN`». Dla wątków mamy podobną sytuację – porównaj zachowanie wątków **przyczepionych** (ang. *attached*) i **odczepionych** (ang. *detached*).

[1] `http://pages.cs.wisc.edu/~remzi/OSTEP/threads-intro.pdf`
[2] `http://pages.cs.wisc.edu/~remzi/OSTEP/threads-api.pdf`
[3] `http://pages.cs.wisc.edu/~remzi/OSTEP/threads-events.pdf`
[4] `https://people.eecs.berkeley.edu/~sseshia/fmee/lectures/TemporalLogicIntro.pdf`

Zadanie 5. Implementacja wątków POSIX skomplikowała semantykę niektórych zachowań procesów, które omawialiśmy do tej pory. Co nieoczekiwanego może się wydarzyć w wielowątkowym procesie uniksowym gdy:

- jeden z wątków zawoła funkcję `fork(2)` lub `execve(2)` lub `_exit(2)`?
- proces zadeklarował procedurę obsługi sygnału «SIGINT», sterownik terminala wysyła do procesu «SIGINT» – który wątek obsłuży sygnał?
- określono domyślną dyspozycję sygnału «SIGPIPE», a jeden z wątków spróbuje zapisać do rury `pipe(2)`, której drugi koniec został zamknięty?
- czytamy w wielu wątkach ze pliku zwykłego korzystając z tego samego deskryptora pliku?

Ściągnij ze strony przedmiotu archiwum «so20_lista_10.tar.gz», następnie rozpakuj i zapoznaj się z dostarczonymi plikami. **UWAGA!** Można modyfikować tylko te fragmenty programów, które zostały oznaczone w komentarzu napisem «TODO».

Zadanie 6. Program «echoclient-thread» wczytuje plik tekstowy zadany z wiersza poleceń i dzieli go na linie zakończone znakiem '\n'. Następnie startuje podaną liczbę wątków, z których każdy wykonuje w pętli: nawiązanie połączenia, wysłanie «ITERMAX» losowych linii wczytanego pliku, zamknięcie połączenia. Wątki robią to tak długo, aż do programu nie przyjdzie sygnał «SIGINT».

Twoim zadaniem jest tak uzupełnić kod, by program uruchamił «nthreads» wątków i poczekał na ich zakończenie. Czemu nie można go łatwo przerobić w taki sposób, żeby główny wątek czekał na zakończenie pozostałych wątków tak, jak robi się to dla procesów przy pomocy `wait(2)`?

Zadanie 7. Program «echoserver-poll» implementuje serwer usługi «echo» przy pomocy wywołania `poll(2)`, ale bez użycia wątków i podprocesów. W kodzie wykorzystano wzorzec projektowy **pętli zdarzeń** (ang. *event loop*) do współbieżnej obsługi wielu połączeń sieciowych. Program «echoserver-select» robi to samo, ale przy pomocy wywołania `select(2)`.

Twoim zadaniem jest uzupełnienie pliku źródłowego «echoserver-poll.c». W pętli zdarzeń należy obsłużyć połączenia przychodzące, nadejście nowych danych na otwartych połączeniach i zamknięcie połączenia. Do przetestowania serwera użyj programu «echoclient-thread» z poprzedniego zadania.

Zadanie 8. Dla jakich parametrów wiersza poleceń obserwujesz niepoprawne zachowanie programów «bug-1» i «bug-2»? Przeanalizuj kod źródłowy programów i wytypuj przyczyny błędów. Następnie zweryfikuj swoje podejrzenia: skompiluj programy z opcją «-fsanitize=thread» i uruchom je. Wyjaśnij znaczenie komunikatów diagnostycznych pochodzących z Thread Sanitizer[5], po czym pokaż jak naprawić błędy.

[5] https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.308.3963