Systemy operacyjne

Lista zadań nr 3

Na zajęcia 24 października 2024

Należy przygotować się do zajęć czytając następujące materiały:

- [1, 7.10, 10.10, 10.15, 10.21, 18.1 – 18.2, 18.6]
- The TTY demystified[1]

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

Zadanie 1. Zaprezentuj sytuację, w której proces zostanie **osierocony**. Uruchom powłokę w nowej instancji emulatora terminala przy pomocy polecenia «`xterm -e 'bash -i'`». W nowej powłoce wystartuj «`sleep 1000`» jako **zadanie drugoplanowe** i sprawdź, kto jest jego rodzicem. Poleceniem «`kill`» wyślij sygnał «`SIGKILL`» do uruchomionej wcześniej powłoki i sprawdź, kto stał się nowym rodzicem procesu «`sleep`». Zauważ, że powłoka jest **liderem sesji**. Co się dzieje z **sesją**, która utraci **terminal sterujący**? Przeprowadź eksperyment wysyłając «`SIGKILL`» do emulatora terminala zamiast do powłoki.

Wskazówka: Obserwuj interakcję systemu z powłoką przy pomocy polecenia «`strace -e trace=signal -p ...`».

Zadanie 2. Zapoznaj się z paragrafami „*Canonical Mode Input Processing*", „*Special Characters*" i „*Writing Data and Output Processing*" podręcznika `termios(4)`. Jak zachowuje się sterownik terminala działającego w **trybie kanonicznym**? Posługując się rysunkiem [1, 62-1] wytłumacz w jaki sposób przetwarza on znaki (w tym kody sterujące) wchodzące do **kolejki wejściowej** i **kolejki wyjściowej**. Jak konfigurację terminala powinien zmienić program na czas wpisywania hasła przez użytkownika? Czemu edytory takie jak `vi(1)` konfigurują sterownik terminala do pracy w trybie niekanonicznym?

Zadanie 3. Wyświetl konfigurację terminala przy pomocy polecenia «`stty -a`». Wskaż znaki, które sterownik terminala: zamienia na sygnały związane z **zarządzaniem zadaniami**, służą do **edycji wiersza**. Program może zostać poinformowany o zmianie **rozmiaru okna** terminala. W tym celu musi zainstalować procedurę obsługi sygnału – którego? Jaką procedurą można wczytać nowy rozmiar okna?

Zadanie 4. Urządzenie terminala zajmuje się interpretacją znaków i sekwencji sterujących[2] przychodzących od sterownika terminala, oraz przetwarzaniem zdarzeń od użytkownika, które zostają zamienione na znaki lub sekwencje znaków, a następnie wysłane do sterownika terminala. Posługując się poleceniem «`echo -e 'sterujacy'; read`» zaprezentuj działanie znaków sterujących oraz sekwencji «`CSI`». Uruchom polecenie «`cat`» i sprawdź jak zachowuje się naciśnięcie klawiszy funkcyjnych i kursora. Czemu w tym przypadku zachowanie programu «`cat`» jest inne niż powłoki poleceń?

Zadanie 5. Mając na uwadze bieżącą konfigurację sterownika terminala wykonaj następujące polecenia:

1. Wstrzymaj zadanie pierwszoplanowe «`sleep 1000`» i przy pomocy wbudowanego polecenia powłoki «`bg`» przenieś to zadanie do wykonania w tle. Jaki sygnał został użyty do wstrzymania zadania?
2. Uruchom «`find /`». W trakcie jego działania naciśnij na przemian kilkukrotnie kombinację klawiszy «`CTRL+S`» oraz «`CTRL+Q`». Czemu program zatrzymuje się i wznawia swoją pracę, skoro sterownik terminala nie wysyłał do niego żadnych sygnałów?
3. Uruchom w powłoce «`bash`» polecenie «`cat - &`». Czemu zadanie zostało od razu wstrzymane? Jaki sygnał otrzymało? Zakończ to zdanie wbudowanym poleceniem powłoki «`kill`».
4. Porównaj działanie polecenia «`cat /etc/shells &`» przed i po zmianie konfiguracji terminala poleceniem «`stty tostop`». Jaki efekt ma włączenie flagi «`tostop`» na zachowanie sterownika terminala?
5. Wykonaj polecenie «`stty -echoctl`». Wyjaśnij co zmieniło się w konfiguracji terminala i zaprezentuj na przykładzie programu «`cat`» pokaż jak zmieniło się przetwarzanie znaków sterujących.

[1] `https://www.linusakesson.net/programming/tty/`
[2] `https://en.wikipedia.org/wiki/ANSI_escape_code`

Ściągnij ze strony przedmiotu archiwum «so21_lista_3.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. Procedury setjmp(3) i longjmp(3) z biblioteki standardowej języka C służą do wykonywania nielokalnych skoków. Uproszczone odpowiedniki tych procedur znajdują się w pliku «libcsapp/Setjmp.s», a definicja «Jmpbuf» w pliku «include/csapp.h». Wyjaśnij co robią te procedury, a następnie przeprowadź uczestników zajęć przez ich kod. Dlaczego «Jmpbuf» nie przechowuje wszystkich rejestrów procesora? Czemu «Longjmp» zapisuje na stos wartość przed wykonaniem instrukcji «ret»?

Zadanie 7. Uzupełnij program «game» tj. prostą grę w szybkie obliczanie sumy dwóch liczb. Zadaniem procedury «readnum» jest wczytać od użytkownika liczbę. Jeśli w międzyczasie przyjdzie sygnał, to procedura ma natychmiast wrócić podając numer sygnału, który przerwał jej działanie. W przeciwnym przypadku zwraca zero i przekazuje wczytaną liczbę przez pamięć pod wskaźnikiem «num_p». Twoja implementacja procedury «readnum» musi wczytać cały wiersz w jednym kroku. Należy wykorzystać procedury siglongjmp(3), sigsetjmp(3) i alarm(2). Pamiętaj, żeby po wczytaniu ciągu znaków zakończyć go znakiem NUL i wyłączyć czasomierz! Kiedy Twój program będzie zachowywać się poprawnie zamień procedury **nielokalnych skoków** na longjmp(3) i setjmp(3). Czemu program przestał działać?

UWAGA! We FreeBSD i macOS zamiast «longjmp» i «setjmp» należy użyć odpowiednio «_longjmp» i «_setjmp».

Zadanie 8. Program «coro» wykonuje trzy współprogramy[3] połączone ze sobą w potok bez użycia pipe(2). Pierwszy z nich czyta ze standardowego wejścia znaki, kompresuje białe znaki i zlicza słowa. Drugi usuwa wszystkie znaki niebędące literami. Trzeci zmienia wielkość liter i drukuje znaki na standardowe wyjście.

W wyniku wykonania procedury «coro_yield» współprogram przekazuje niezerową liczbę do następnego współprogramu, który otrzyma tę wartość w wyniku powrotu z «coro_yield». Efektywnie procedura ta implementuje **zmianę kontekstu**. Taką prymitywną formę **wielozadaniowości kooperacyjnej** (ang. *cooperative multitasking*) można zaprogramować za pomocą setjmp(3) i longjmp(3).

Zaprogramuj procedurę «coro_switch» tak, by wybierała następny współprogram do uruchomienia i przełączała na niego kontekst. Jeśli współprogram przekazał wartość parametru «EOF», to należy go usunąć z listy aktywnych współprogramów.

Program używa listy dwukierunkowej «TAILQ» opisanej w queue(3). Zmienna «runqueue» przechowuje listę aktywnych współprogramów, «running» bieżąco wykonywany współprogram, a «dispatcher» kontekst programu, do którego należy wrócić, po zakończeniu wykonywania ostatniego aktywnego współprogramu.

Literatura

[1] „*Advanced Programming in the UNIX Environment*”
W. Richard Stevens, Stephen A. Rago;
Addison-Wesley Professional; 3rd edition; 2013

[2] „*The Linux Programming Interface: A Linux and UNIX System Programming Handbook*”
Michael Kerrisk; No Starch Press; 1st edition; 2010

[3] https://en.wikipedia.org/wiki/Coroutine