

Systemy komputerowe

Pracownia nr 2

Termin oddawania 20 czerwca 2019

UWAGA! Przeczytaj dokładnie poniższy tekst zanim przystąpisz do rozwiązywania zadań!

Głównym podręcznikiem do zajęć praktycznych jest „The Linux Programming Interface: A Linux and UNIX System Programming Handbook”. Należy zapoznać się z treścią §2 w celach poglądowych, a pozostałe rozdziały czytać w razie potrzeby. Bardziej wnikliwe wyjaśnienia zagadnień można odnaleźć w książce „Advanced Programming in the UNIX Environment”. Zanim sięgniesz do zasobów Internetu zapoznaj się z odpowiednimi stronami podręcznika systemowego poleceniami «man» i «apropos».

Rozwiązania mają być napisane w języku C (a nie C++). Kompilować się bez błędów i ostrzeżeń (opcje «-std=gnu11 -Wall -Wextra») kompilatorem gcc lub clang pod systemem Linux. Do rozwiązań musi być dostarczony plik Makefile, tak by po wywołaniu polecenia «make» otrzymać pliki binarne, a polecenie «make clean» powinno zostawić w katalogu tylko pliki źródłowe. Rozwiązania mają być dostarczone poprzez system oddawania zadań na stronie zajęć.

Należy użyć szkieletów rozwiązań udostępnionych na stronie przedmiotu. W komentarzu na początku pliku należy wpisać swoje imię, nazwisko oraz numer indeksu, jednocześnie oświadczając, że jest się jedynym autorem kodu źródłowego. Należy odpowiedzieć na wszystkie pytania zadane w komentarzu – będziemy je czytać i uwzględniać przy przydzielaniu punktów. Ocena poprawności rozwiązań nie może wymagać od sprawdzającego czynności innych niż uruchomienie programu! Na wyjście programu należy wydrukować komunikaty niezbędne do przekonania sprawdzającego, że program robi to co powinien.

Zadanie 1 (2). Napisz program, który utworzy proces **zombie**. Należy wskazać nieumarły proces potomny uruchamiając polecenie «ps» z użyciem wywołań **fork(2)** i **execve(2)**, a następnie zakończyć cały program. Jeśli użytkownik przekaze parametr «--bury» do programu, to należy zapobiec powstawaniu zombie, tj. zignorować sygnał SIGCHLD z użyciem **sigaction(2)**.

Zadanie 2 (2+1). Napisz program, który utworzy proces **sierotę**. Niech proces główny przyjmie rolę **źniwiarza** (ang. *reaper*) przy użyciu **prctl(2)**. Używając wywołania **fork(2)** utwórz kolejno syna i wnuka. Następnie osieroć wnuka zabijając syna. Uruchom polecenie «ps», aby wskazać kto przygarnął sierotę. Punkt **bonusowy** zostanie przydzielony tylko wtedy, gdy program będzie wolny od potencjalnych wyścigów. Prawidłowe rozwiązanie nie może polegać na arbitralnie dobranych przez programistę opóźnieniach.

Wskazówka: Rozważ użycie grup procesów, przeczytaj podręczniki **setpgrp(2)**, **waitpid(2)**, **killpg(2)**.

Zadanie 3 (3). Napisz program, który wygeneruje błąd odwołania do pamięci. Obsłuż sygnał SIGSEGV z pomocą **sigaction(2)**. Zinterpretuj dane zawarte w drugim «siginfo_t» i trzecim argumentem «ucontext_t» procedury obsługi sygnału. Wypisz na «stderr» komunikat zawierający informacje o:

- adresie powodującym błąd odwołania «si_addr»,
- typie błędu «si_code» w postaci czytelnej dla człowieka, tj. MAPERR lub ACCERR,
- adresie wierzchołka stosu i adresie instrukcji powodującej błąd «uc_mcontext».

... i wydrukuj **ślad wywołań** procedurą **backtrace_symbols_fd(3)**, po czym zakończ działanie programu.

Twój program ma przetestować następujące usterki w zależności od podanego parametru:

- «--maperr» odczyt z nieodwzorowanej pamięci,
- «--accerr» zapis do pamięci tylko do odczytu.

Wygenerowanie powyższych usterek nie może używać adresów arbitralnie wybranych przez programistę. Obszar adresów o pożądanych właściwościach należy utworzyć z użyciem funkcji **mmap(2)** i **unmap(2)**.

UWAGA! Użycie funkcji, która nie jest wielobieźna, w procedurze obsługi sygnału jest poważną usterką w programie.