Etap	1	2	3	4	Suma
Punkty	4	8	7	6	25
Wynik					

L2: Notae Caesarianae

Jest rok 55 p.n.e., sytuacja wojny Republiki Rzymskiej przeciwko plemionom celtyckim w Galii jest dynamiczna. Niestety, jak się okazało, galijskie plemiona nauczyły się w od początku wojny czytać teksty łacińskie i greckie. Korespondencja prowadzona przez Julisza Cezara z senatem, a w szczególności z konsulami Pompejuszem i Crassusem, jest więc zagrożona w wypadku złapania posłańców. Aby tego uniknąć, korespondencja będzie teraz szyfrowana zaawansowanym algorytmem kryptograficznym, tak zwanym szyfrem Cezara. Jednak ze względu na ilość korespondencji, używane przez legiony rzymskie jednoprocesowe algorytmy nie nadążają z szyfrowaniem.

Na szczęście wraz z nową dostawą zapasów do Cezara dotarły nowe, wielordzeniowe serwery. Żeby jednak wykorzystać ich potencjał, jednoprocesowy program szyfrujący musi zostać przepisany na program wieloprocesowy.

Twoim zadaniem jest napisanie programu, który przy pomocy wielu procesów, będzie równolegle szyfrował przekazywaną do Rzymu korespondencję. Program startowy został umieszczony w repozytorium. Twoje rozwiązanie powinno być na nim bazowane. **Pamiętaj o sprawdzaniu błędów zgłaszanych przez wywoływane funkcje oraz o zwalnianiu zaalokowanych zasobów takich jak pamięć i deskryptory.** Razem z początkowym kodem dodany jest też plik `bellum` z przykładowym tekstem do szyfrowania.

Etapy:

1. [4 p.] Program przyjmuje parametry wejściowe opisane przez funkcję `usage` w startowym programie. Po uruchomieniu programu uruchom $0 < k < 8$ procesów potomnych. Sprawdź, czy k jest w tym zakresie. Każdy z procesów (rodzic i dzieci) powinien wypisać na ekran swój PID, a następnie zakończyć pracę. Proces rodzic czeka na zakończenie pracy wszystkich procesów dzieci, po czym sam się kończy.

2. [8 p.] Otwórz plik wskazany przez parametr p. Użyj w tym celu niskopoziomowego API (`open`, `close`, itd.). Czytanie zawartości pliku przeplataj z uruchamianiem nowych procesów potomnych tak, aby i-ty proces miał dostęp do i-tego fragmentu pliku. Zawartość plików dzielona jest po równo. Jeśli nie jest to możliwe, to ostatni proces może mieć kilka znaków mniej lub więcej do przetworzenia.

   Kiedy wszystkie procesy potomne będą już uruchomione, wyślij do wszystkich sygnał `SIGUSR1`. Po jego otrzymaniu, procesy potomne się kończą.

   Pamiętaj aby poprawnie obsłużyć błąd `EINTR` zgłaszany przez funkcje systemowe (t.j. nie przerywaj programu!).

3. [7 p.] Każdy proces potomny po otrzymaniu sygnału `SIGUSR1` rozpoczyna szyfrowanie. Przebiega w pętli po swoim fragmencie pliku, dodając do wartości ASCII każdego znaku 3. W miarę szyfrowania nowa treść zapisywana jest do pliku `<p>-<i>`, gdzie i jest numerem procesu potomnego (nie jego PID!). Można założyć, że w pliku p znajdują się tylko małe litery. Zamiana ma być cykliczna, tzn. `z` zamienione powinno być na `c` itd. Między zamianami kolejnych znaków proces śpi $0,1$ sekundy. Poprawnie obsłuż przypadek przerwania snu sygnałem (tzn. trzeba „dospać" nieprzespany czas). Po zapisaniu danych do pliku proces kończy pracę.

4. [6 p.] Dodaj poprawną obsługę sygnału `SIGINT` w procesie rodzicu i w procesach dzieci. Proces dziecka po otrzymaniu `SIGINT` powinien zamknąć wszystkie deskryptory, wysłać `SIGINT` do wszystkich innych procesów w grupie (w tym do rodzica) i się zakończyć. Proces rodzica powinien wysłać `SIGINT` do dzieci i czekać na ich zakończenie, a następnie samemu się zakończyć. Z każdym razem gdy którykolwiek proces otrzyma `SIGINT`, wypisz informację o tym na `STDOUT`. Nie rób tego w funkcji obsługującej sygnał (ang. signal handler)!