SPIS TREŚCI

1. AIO
   - Opis
   - Struktura aiocb
2. Funkcje
   - $aio\_read()$
   - $aio\_write()$
   - $aio\_fsync()$
   - $aio\_error()$
   - $aio\_return()$
   - $aio\_suspend()$
   - $aio\_cancel()$

1. AIO (man 7)

- Opis
  - POSIX pozwala aplikacjom inicjalizować jedną lub więcej asynchronicznych operacji wejścia/wyjścia
  - proces nie czeka aż operacja się zakończy
  - o zakończeniu operacji proces jest informowany przez :
    - → dostarczenie sygnału
    - → utworzenie instancji wątku
    - → lub w ogóle nie jest informowany
- Struktura aiocb
  - nazwa od „asynchronous I/O control block”
  - $\#include\ <aiocb.h>$
  - pola :
    - → *int* *aio_fildes* → deskryptor pliku do operacji I/O
    - → *off_t* *aio_offset* → przesunięcie z którego wykonywana jest operacja I/O
    - → *volatile void* **aio_buf* → bufor używany do transferu danych dla operacji czytania/pisania
    - → *size_t* *aio_nbytes* → długość bufora buf
    - → *int* *aio_reqprio* → priorytet żądania, wartość odejmowana od wywołującego wątku w celu określenia priorytetu wykonania żądania wejścia/wyjścia
      → wartość musi być z przedziału od 0 do wartości zwracanej przez sysconf(SC_AIO_PRIO_DELTA_MAX)
    - → *struct sigevent* *aio_sigevent* → struktura określająca jak wywołujący funkcję proces ma zostać poinformowany, kiedy asynchroniczna funkcja I/O zostanie zakończona
      → możliwe wartości : SIGEV_NONE, SIGEV_SIGNAL, SIGEV_THREAD
    - → *int* *aio_lio_opcode* → typ operacji, która ma być wykonana (używany tylko dla lio_listio)

2. Funkcje

- $aio\_read()$ $[\#include\ <aio.h>]$
  - **sygnatura :**
    int aio_read (struct aiocb *aiocbp)
  - **działanie funkcji :**
    → asynchroniczny odczyt
    → funkcja zleca asynchroniczną operację „n=read(fd,buf,count)" z argumentami kolejno :
    ☞ fd = aiocbp−> fildes
    ☞ buf = aiocbp −> aio_buf
    ☞ count = aiocbp −> aio_nbytes
    → zatem z pliku o deskryptorze $aiocbp{-}>\ fildes$ jest przepisywane $aiocbp{-}>\ aio\_nbytes$ bajtów do bufora $aiocbp \rightarrow aio\_buf$
    → status zakończenia operacji może być pobrny po jej zakończeniu przez $aio\_return()$
    → dane odczytywane począwszy od przesunięcia $aiocbp{-}>\ aio\_offset$ niezależnie od aktualnej pozycji w pliku
    → wywołanie powraca zaraz po tym, jak operacja zostanie zakolejkowana
    ☞ odczyt może, ale nie musi być ukończony, kiedy funkcja powraca
    ☞ można sprawdzić, czy operacja jest zakończona wywołując $aio\_error()$
    → z plików regularnych nie są czytane dane poza ich maksymalnym offsetem
    → dobrym pomysłem jest wyzerowanie bloku kontrolnego $aiocbp$ przed użyciem
    ☞ blok ten nie może być zmieniany podczas odczytu
  - **argumenty**
    → struct aiocb * aiocbp → blok kontrolny operacji asynchronicznej (**pola**)
  - **wartość zwracana**
    → 0 → jeśli operacja wczytywania asynchronicznego rozpoczęła się pomyślnie
    → -1 → jeśli operacja wczytywania asynchronicznego nie rozpoczęła się pomyślnie
    ☞ wówczas zlecenie nie jest kolejkowane
    ☞ errno ustawiane na kod błędu
    (i) EGAIN – brak zasobów

- $aio\_write()$ $[\#include < aio.h >]$
  - **sygnatura :**
    int aio_write (struct aiocb *aiocbp)
  - **działanie funkcji**
    → asynchroniczny zapis
    → zapisuje do pliku o deskryptorze $aiocbp{-}>\ fildes$ blok danych o wielkości $aiocbp \rightarrow aio\_nbytes$ z bufora $aiocbp \rightarrow aio\_buf$
    → status zakończenia operacji możliwy do pobrania przez $aio\_return()$
    → jeśli O_APPEND nie jest ustawione → dane będą zapisywane od $aiocbp \rightarrow aio\_offset$
    → jeśli O_APPEND ustawione → dane dopisywane na końcu pliku
    → wywołana funkcja powraca zaraz po zakolejkowaniu zlecenia asynchronicznej operacji I/O
    ☞ odczyt może ale nie musi być ukończony, gdy funkcja wraca
    ☞ można sprawdzić czy odczyt ukończony wywołując $aio\_error()$
  - **argumenty :**
    → $struct\ aiocb\_aiocbp$ → blok kontrolny operacji asynchronicznej (**pola**)
  - **wartość zwracana**
    → 0 → jeśli operacja zapisu asynchronicznego rozpoczęła się pomyślnie
    → -1 → jeśli operacja zapisu asynchronicznego nie rozpoczęła się pomyślnie
    ☞ wówczas zlecenie niekolejkowane
    ☞ errno ustawiane na kod błędu

- $aio\_fsync()$ $[\#include < aio.h >]$
  - **sygnatura :**
    $int\ aio\_fsync\ (int\ op, struct\ aiocb * aiocbp)$
  - **działanie funkcji :**
    → asynchroniczna synchronizacja pliku
    → funkcja realizuje operację synchronizacji na wszystkich zalegających operacjach I/O powiązanych z plikiem o deskryptorze $aiocbp{->}\ fildes$
    → funkcja używa pola $aiocbp{->}\ aio\_sigevent$
  - **argumenty**
    → $int\ op$ → opcja synchronizacji
    ☞ O_SYNC → wszystkie aktualnie zakolejkowane operacje I/O powinny być zakończone jak w wywołaniu f_sync()
    ☞ O_DSYNC → wywołanie jest asynchroniczną analogią dla fdatasync()
    → $struct\ aiocb\ *aiocbp$ → blok kontrolny operacji asnchronicznej (**pola**)
  - **wartość zwracana**
    → 0 → jeśli operacja synchronizacji rozpoczęła się pomyślnie
    → -1 → jeśli operacja synchornizacji nie rozpoczęła się pomyślnie
    ☞ ustawia errno na kod błędu

- $aio\_error()$ $[\#include < aio.h >]$
  - **sygnatura :**
    $int\ aio\_error(\ const\ struct\ aiocb\ *aiocbp\ )$
  - **działanie funkcji**
    → zwraca status błędu asynchronicznej operacji I/O
  - **argumenty :**
    → $const\ struct\ aiocb\ *aiocbp$ → blok kontrolny zlecenia
  - **wartość zwracana :**
    → $EINPROGRESS$ → jeśli zlecenie nie zostało jeszcze ukończone
    → $ECANCELED$ → jeśli zlecenie zostało przerwane
    → 0 → gdy zlecenie poprawnie zakończone
    → w innych przypadkach kod błędu
    ☞ EINVAL → gdy $aiocbp$ nie wskazuje na blok kontrolny asynchronicznego zlecenia I/O

- $aio\_return()$ $[\#include < aio.h >]$
  - **sygnatura :**
    $ssize\_t\ aio\_return(\ struct\ aiocb\ *aiocbp\ )$
  - **działanie funkcji**
    → pobiera status zakończenia asynchronicznej operacji I/O
    → funkcja zwraca ostateczny status zakończenia zlecenia wskazywanego przez $aiocbp$
    → powinna być wywołana tylko raz dla danego zlecenia, gdy $aio\_error()$ zwróci wartość różną od $EINPROGRESS$
    ☞ jeśli zlecenie nie było zakończone to zachowanie funkcji niezdefiniowane
  - **argumenty :**
    → $struct\ aiocb\ *aiocbp$ → blok kontrolny asynchronicznej operacji I/O
  - **wartość zwracana :**
    → wartość która byłaby zwrócona w przypadku synchronicznego wywołania operacji I/O $(read(), write()$ lub $fsync()$)
    → w przypadku błędu zwracana wartość błędu

- $aio\_suspend()$ $[\#include < aio.h >]$
  - **sygnatura :**
    $int\ aio\_suspend\ (const\ struct\ aiocb\ *const\ list[], int\ nent, const\ struct\ timespec\ *timeout)$
  - **działanie funkcji :**
    → czekanie na zakończenie asynchronicznej operacji I/O lub określony zadany czas $(timeout)$
    → funkcja wstrzymuje wywołujący wątek do chwili aż co najmniej jedno asynchroniczne zlecenie I/O w liście $list$ o długości $nent$ zostanie ukonczone, dostarczony będzie sygnał lub upłynie $timeout$
    → jeśli element na liście jest NULL to jest ignorowany
  - **argumenty :**
    → const struct aiocb * list → tablica bloków kontrolnych asynchronicznych operacji I/O
    → int nent → wielkość tablicy $list$
    → const struct timespec *timeout → jeśli nie jest NULL, to jest to maksymalny czas oczekiwania
  - **wartość zwracana :**
    → 0 → gdy zakończyła się któraś z operacji w liście $list$
    → -1 → gdy funkcja zakończy się, a żadna z operacji w liście $list$ jeszcze się nie zakończyła
    ☞ EAGAIN → jeśli minął $timeout$ a żadna z operacji w $list$ się nie zakończyła
    ☞ EINTR → funkcja przerwana przez sygnał

- $aio\_cancel()$ $[\#include < aio.h >]$
  - **sygnatura :**
    $int\ aio\_cancel\ (int\ fildes, struct\ aiocb\ *aiocbp)$
  - **działanie funkcji :**
    → przerwanie zaległej asynchronicznej operacji I/O
    → funkcja stara się przerwać zalegające asynchroniczne zlecenia I/O dla podanego deskryptora $fildes$
    → jeśli $aiocbp$ jest NULL to przechwytywane będą wszystkie zlecenia
    ☞ w przeciwnym przypadku przerywane będzie jedynie zlecenie określone przez blok kontrolny $aiocbp$
    → dla przerwanych zleceń zgloszone będzie normalne asynchroniczne powiadomienie
    ☞ zlecenia zwrócą status ustawiony na -1 i status błędu ECANCELED
    → blok konrolny zlecenia, które nie mogło być przerwane nie jest modyfikowany
    → jeśli $aiocbp$ nie jest NULL, a $fildes$ różni się od $aiocbp{->}\ aio\_fildes$ to działanie funkcji jest nieokreślone
  - **argumenty :**
    → $int\ fildes$ → deskryptor pliku, dla którego chcemy anulować asynchroniczne operacje I/O
    → $struct\ aiocb\ *\ aiocbp$ → opcjonalny blok kontrolny, określający konkretne zlecenie, które ma być przerwane (jeśli NULL, to wszystkie o deskryptorze $fildes$) (**pola**)
  - **wartość zwracana :**
    → AIO_CANCELED → jeśli wszystkie zlecenia I/O zostały przerwane
    → AIO_NOTCANCELED → jeśli przynajmniej jedna z operacji nie została przerwana, bo jest w trakcie wykonania
    ☞ można wtedy sprawdzić status poszczególnych zleceń używając **aio_error()**
    → AIO_ALLDONE → jeśli wszystkie operacje całkowicie zrealizowane przed wywołaniem
    → -1 → w innych przypadkach
    ☞ ustawiane errno