1- Explicar o objetivo e os parâmetros das funções:

-> getContext(&context)

Inicializa uma estrutura do tipo ucontext_t e aponta a variável &context para a mesma.

-> setContext(&context)

Troca imediatamente o contexto atual para o contexto apontado por &context

-> swapContext(&current, &another)

Salva o contexto atual em &current e então troca o contexto para o que está contido em &another.

-> makeContext(&context, func(), arg_qnt, args)

Salva no contexto de &context uma função representado pelo parâmetro func(), indicando que ela possui uma quantidade de argumentos igual o valor da variável 'arg_qnt' sendo que os argumentos de fato são passados utilizando a variável 'args'.

Essa função será executada após uma chamada da função swapContext() que troca um contexto qualquer pelo contexto contido em &context ou uma chamada a setContext() passando como parametro o contexto &context.

2 - Explicar o significado de cada um dos campos da estrutura ucontext_t que foram utilizados no código:

3 - Explicar em detalhes o funcionamento do código do arquivo contexts.c:

-> No inicio da funcao main() temos um print da string "Main INICIO" que evidencia o começo da execução do código.

-> Chamada a função getContext() passando por parâmetro a variável contextPing. Isso inicializa uma estruta do tipo ucontext_t e aponta contextPing para a mesma.
-> Alocando uma stack de tamanho constante (definido por STACKSIZE)
-> Caso a stack tenha sido alocada sem problemas: Guarda-se no contexto que essa stack deve ser utilizada, o tamanho dessa stack, define que não existem flags e que não se estabeleceu nenhum processo sucessor após o término da execução de um processo que utilize esse contexto.
-> Uma chamada a função makeContext(), indicando que a execução de uma chamada a swapContext() ou setContext() que passe como parâmetro contextPing deve trigar a execução do código contido em bodyPing() passando somente 1 parâmetro, a string "PING"

-> Chamada a função getContext() passando por parâmetro a variável contextPong. Isso inicializa uma estruta do tipo ucontext_t e aponta contextPong para a mesma.
-> Alocando uma stack de tamanho constante (definido por STACKSIZE)

-> Caso a stack tenha sido alocada sem problemas: Guarda-se no contexto que essa stack deve ser utilizada, o tamanho dessa stack, define que não existem flags e que não se estabeleceu nenhum processo sucessor após o término da execução de um processo que utilize esse contexto.
-> Uma chamada a função makeContext(), indicando que a execução de uma chamada a swapContext() ou setContext() que passe como parâmetro contextPong deve trigar a execução do código contido em bodyPong() passando somente 1 parâmetro, a string “PONG”

-> Um chamada a função swapContext(). O contexto atual de mainContext() é salvo e então deve ser trocado por ‘contextPing’. O contexto de contextPing é carregado na memória e ,como definido anteriormente, é executado a funcao bodyPing()

-> bodyPing() executa uma chamada a swapContext() que salva o contexto atual de contextPing e troca o contexto para contextPong. Como definido anteriormente isso triga a execução de bodyPong().

-> bodyPong() executa uma chamada a swapContext() que salva o contexto atual em contextPong e troca o contexto para contextPing. Como definido anteriormente isso triga a execução de bodyPing()

As chamadas a função swapContext() e consequentemente a execução de bodyPing() e bodyPong() serão realizadas em loop uma quantidade de vezes igual ao valor da variável ‘i’ contida em cada uma delas

4 - Diagrama de tempo de execução: